ARL-TR-7469 ● SEP 2015

US Army Research Laboratory

KMCThinFilm: A C++ Framework for the Rapid Development of Lattice Kinetic Monte Carlo (kMC) Simulations of Thin Film Growth

by James J Ramsey

Approved for public release; distribution is unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-7469 ● SEP 2015

US Army Research Laboratory

KMCThinFilm: A C++ Framework for the Rapid Development of Lattice Kinetic Monte Carlo (kMC) Simulations of Thin Film Growth

by James J Ramsey
Computational and Information Sciences Directorate, ARL

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
Sep 2015	Final	NA

4. TITLE AND SUBTITLE
KMCThinFilm: A C++ Framework for the Rapid Development of Lattice Kinetic Monte Carlo (kMC) Simulations of Thin Film Growth

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
James J Ramsey

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIH-C
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-7469

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A new C++ library for the rapid development of lattice-based kinetic Monte Carlo (kMC) simulation codes, KMCThinFilm, has been introduced. This library implements versions of the Bortz-Kalos-Lebowitz (BKL) algorithm for choosing sequences of random events on a lattice in a kMC simulation, and these implementations have been optimized so that only a subset of the cells of the lattice has to be scanned at each time step of the simulation. Researchers who wish to implement kMC models of some physical processes can use this library for the BKL algorithm rather than program an ad hoc and possibly naïve implementation of it, and focus on writing the code needed to determine the propensities for the possible events in their kMC model and the means of executing them. Example applications, one of a patterned substrate and one of ballistic deposition of a pseudo-polycrystalline film, demonstrate the flexibility of the library. The KMCThinFilm library is also capable of approximate parallel kMC simulations, and its parallel efficiency appears to be near the same or better than a previous parallel implementation described in a 2007 article by F Shi, Y Shim, and JG Amar, "Parallel Kinetic Monte Carlo Simulations of Two-Dimensional Island Coarsening," Physical Review E, volume 76, 031607. However, parallel scalability for simulations on smaller lattices tends to be poor, so it is recommended that parallelism only be used for simulations that are too computationally intensive to run on a workstation.

15. SUBJECT TERMS
kinetic Monte Carlo, C++, software

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			James J Ramsey
Unclassified	Unclassified	Unclassified	UU	52	19b. TELEPHONE NUMBER (Include area code) 410-278-5614

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures iv

List of Tables vii

1. Introduction 1

2. Concepts and Algorithms of the "KMCThinFilm" Library 3

3 Example Applications 12

3.1 Patterned Substrate 12

3.2 Pseudo-polycrystal Formed by Ballistic Deposition 18

4. Parallel Scalability 27

5. Conclusions 35

6. References and Notes 36

List of Symbols, Abbreviations, and Acronyms 40

Distribution List 42

List of Figures

Fig. 1 Topmost planes of a lattice whose indices along the i and j directions are in the interval [0, i_{max}] and [0, j_{max}], respectively. The i-, j-, and k-axes are not necessarily orthogonal. 4

Fig. 2 Diagram of cells in a simple cubic lattice affected by an event with offsets (−1,0,0), (+1,0,0), and (0,+1,0) (in addition to the always present zero offset) that changes the cells (a_1,b,c) and (a_2,b,c) 6

Fig. 3 Lattice partitioned across 4 processes according to the compact scheme. Each process is assigned a rank from 0 to 3. The ghost regions along the boundaries of each partition are copies of the lattice cells of a neighboring processor, and in the diagram, the processor from which they are copied is shown via their color. Periodic boundary conditions apply....9

Fig. 4 Lattice partitioned across 4 processes according to the row-based scheme. The ghost regions along the boundaries of each partition are copies of the lattice cells of a neighboring processor, and in the diagram, the processor from which they are copied is shown via their color. Periodic boundary conditions apply. 9

Fig. 5 Sector boundaries for a lattice decomposed across 4 processes according to the compact scheme. Partition boundaries are indicated by thick solid lines, while the sector boundaries are indicated by thinner solid lines. Active sectors are shown in the color corresponding to the rank of the partition to which they belong. The dotted lines show the boundaries of the regions affected by events that occur within the active sectors. The parts of these regions that affect ghost cells are shown in the color corresponding to the ranks of the cells of which the ghost cells are copies. Here, the active sector happens to be the upper left of each partition, but at a given time, the active sector could be in the upper right, lower right, or lower left....10

Fig. 6 Variation of E_s and E_d in patterned substrate models A and B from Kuronen et al. Here $i,j \in [0,352)$ and the size of an individual domain is 22 × 22. For E_s, brighter colors constitute higher values, while for E_d, the reverse is true. 13

Fig. 7 Arrangement of islands at the end of the simulation of model A for temperatures a) T = 340 K, b) T = 390 K, and c) T = 440 K, using the solver based on the Schulze[12] algorithm, and for temperatures d) T = 340 K, e) T = 390 K, and f) T = 440 K, using the binary tree algorithm. The grey level at a point indicates the height of the column of particles there....16

Fig. 8 Arrangement of islands at the end of the simulation of model B for temperatures a) $T = 340$ K, b) $T = 390$ K, and c) $T = 440$ K, using the solver based on the Schulze[12] algorithm, and for temperatures d) $T = 340$ K, e) $T = 390$ K, and f) $T = 440$ K, using the binary tree algorithm. The grey level at a point indicates the height of the column of particles there.....17

Fig. 9 Slice of simulated thin film for flux $F = 0.1$ ML/s and angles of incidence from a) $\theta = 0°$, b) $\theta = 15°$, c) $\theta = 30°$, and d) $\theta = 45°$. Here, the x-axis (not shown) points to the right, and the y-axis (also not shown) points out of the page. 21

Fig. 10 Slice of simulated thin film for flux $F = 0.1$ ML/s and angles of incidence from a) $\theta = 60°$ and b) $\theta = 75°$. Here, the x-axis (not shown) points to the right, and the y-axis (also not shown) points out of the page. 22

Fig. 11 Slice of simulated thin film for flux $F = 0.1$ ML/s and angle of incidence $\theta = 85°$. Here, the x-axis (not shown) points to the right, and the y-axis (also not shown) points out of the page. 22

Fig. 12 Histograms of the growth angle and azimuth (with respect to the negative x-axis) of grains of films grown with a deposition flux of 10 ML/s and angles of incidence varying from 0° to 30°. 23

Fig. 13 Histograms of the growth angle and azimuth (in respect to the negative x-axis) of grains of films grown with a deposition flux of 10 ML/s and angles of incidence varying from 45° to 85°. The mean and standard deviations of the growth angle and azimuth, as well as the growth angle predicted from the theory of Tait et al. are also shown. 24

Fig. 14 Relative density of a simulated film deposited by ballistic deposition versus the angle of incidence. A film with a relative density of 1.0 has no vacancies. Error bars represent a 99.7% confidence interval. 25

Fig. 15 Parallel efficiency versus the number of processing cores for the implementation of an island coarsening model using the "KMCThinFilm" library on a Cray XC40 cluster. The model is applied over domains with in-plane dimensions of a) 256 × 256, b) 512 × 512, c) 1,024 × 1,024, and d) 1,600 × 1,600. The parallel time-stepping scheme here uses a fixed time step set to $1/D_0$. Both row-based and compact decomposition are used with the binary tree and Schulze solvers. For models with in-plane dimensions from 512 × 512 to 1,600 × 1,600, parallel efficiencies from the island coarsening implementation of Shi, Shim, and Amar (running on a different cluster) are also shown. 30

Fig. 16 Wall clock time versus the number of processing cores for the implementation of an island coarsening model using the "KMCThinFilm" library on a Cray XC40 cluster. The model is applied over domains in-plane dimensions of a) 256 × 256, b) 512 × 512, c) 1,024 × 1,024, and d) 1,600 × 1,600. The parallel time-stepping scheme here uses a fixed time step set to $1/D_0$. Both row-based and compact decomposition are used with the binary tree and Schulze solvers. 31

Fig. 17 Effects of the choice of adaptive time-step scheme on the parallel scalability of the implementation of an island coarsening model on a 512 × 512 domain, using the "KMCThinFilm" library on a Cray XC40 cluster. These figures show a) strong parallel scalability, b) wall clock time, c) average value of the adaptive time step, and d) the number of parallel time steps (i.e., the loops over sectors). "Max. Avg. Prop", "Max. Sing. Prop." and "Fixed" represent different parallel time-stepping schemes. These were determined using the solver based on the Schulze algorithm with row-based decomposition. 33

Fig. 18 Effects of the choice of adaptive time-step scheme on the parallel scalability of the implementation of an island coarsening model on a 256 × 256 domain, using the "KMCThinFilm" library on a Cray XC40 cluster. These figures show a) strong parallel scalability, b) wall clock time, c) average value of the adaptive time step, and d) the number of parallel time steps (i.e., the loops over sectors). "Max. Avg. Prop", "Max. Sing. Prop." and "Fixed" represent different parallel time-stepping schemes. These were determined using the solver based on the Schulze algorithm with row-based decomposition. 34

Fig. 19 Close-up of the results of an island coarsening simulation after 10^3 s of simulation time, on a 512 × 512 domain distributed over 8 processing cores, using the maximum average propensity adaptive parallel time-stepping scheme with $n_{stop} = 1$, the Schulze solver, and row-based parallel decomposition. These results show a "shish kebab" effect, where features of the simulation (in this case, particles with no neighbors or only 1 neighbor) are artificially attracted to sector boundaries due to approximations in the parallel kMC method. 35

List of Tables

Table 1 Average wall clock times and average times per event for simulations of model A on an Intel Xeon 2.3-GHz node of a computational cluster.....17

Table 2 Average wall clock times and average times per event for simulations of model B an Intel Xeon 2.3-GHz node of a computational cluster. .18

Table 3 Wall clock time and average times per event spent in simulations with deposition flux F = 10 ML/s. 26

Table 4 Wall clock time and average times per event spent in simulations with deposition flux F = 0.1 ML/s.....27

INTENTIONALLY LEFT BLANK.

1. Introduction

The kinetic Monte Carlo (kMC) method can be used to simulate a system whose evolution over time can be modeled as a chain of random events, where each random event depends only on the current state of the system.[1] In particular, the kMC method has been used to simulate the growth of films,[2–10] where the random events are instances of particles undergoing processes such as deposition, adsorption, desorption, diffusion, and chemical reactions. Typically, a kMC simulation uses an algorithm where at a given time step in the simulation, the set of all possible events and their associated rate constants or "propensities" (i.e., probabilities per unit time) is determined. One of these events is then randomly chosen to be executed, with the choice weighted according to the propensities of the possible events, and then the simulation time is advanced by $-\ln(r)/p_{\mathrm{tot}}$, where r is a random number uniformly chosen from the open interval $(0,1)$, and p_{tot} is the sum of all the propensities of all possible events.[1] To simplify the process of determining the set of possible events, the locations of the particles in the system are usually assumed to be restricted to be at or near the sites of a lattice, and events such as adsorption, diffusion, and so on, are centered near these sites. One can then determine the set of possible events by scanning the lattice and checking what sites are vacant or occupied with particular particle species.

Implementation of the general kMC algorithm described above can be performed with varying degrees of optimization. A simple and straightforward but naïve implementation of a lattice-based kMC simulation scans through all L sites of the lattice at each time step to generate the set of possible events afresh and then performs a linear search on the set of N possible events in order to find one to execute, operations which are $O(L)$ and $O(N)$, respectively. The scaling of the latter operation can be improved to $O(\log_2 N)$[11] or, in some cases, can even be independent of N.[12] If the events in the simulation have only local effects, that is, each event only immediately affects the sites in a small neighborhood near the site where the event is centered, then the scalability of the former operation can be improved by initially generating the set of possible events at the beginning of the simulation, and then, at each time step, only updating the possible events in the set that were affected by the execution of the event chosen at the step. Of course, these optimizations are more complicated to code than a naïve approach and thus present the opportunity to introduce bugs in a new implementation. Ideally, rather than have each researcher who employs a kMC method attempt to program such optimized implementations from scratch, there would be optimized implementations already available that researchers could adopt and apply in their own work. Such is already

the case for other kinds of atomistic numerical simulations, such as molecular dynamics or various ab initio methods.

Nonetheless, kMC simulations have typically been implemented using ad hoc code written for a particular line of research.[13] A few kMC codes are available that are designed to be of more general use, such as "CARLOS,"[14,15] "CHIMP" HIerarchical Modeling Program,[16,17] "MONTY,"[18,19] and Graph Theoretical kMC.[20,21] These take in input files that describe the available possible events of the system and their propensities. The "kmos" project[13,22] is somewhat similar to these, except that it takes in an input file (written in Extensible Markup Language [XML]) in order to generate Fortran 90 code that contains the actual implementation of a kMC simulation. These codes tend to be more optimized than a typical ad hoc kMC code, but are still limited by what they allow as inputs. For example, CARLOS and Graph Theoretical kMC are designed to model reactions on surfaces. The means of specifying possible events in kmos precludes forms of deposition that involve shadowing (where one part of the growing surface blocks overshadows another part and prevents depositing atoms from reaching it). None of these codes would be capable of, for example, accommodating some kind of prepatterning or strain field to induce quantum dots, as in, for example, the work of Kuronen et al.[5] or Kiravittaya and Schmidt.[23] In short, while these codes have some advantages over purely ad hoc code, they may not have the desired flexibility needed by a researcher. In contrast to these codes are KMCLib[24] and Stochastic Parallel PARticle Kinetic Simulator (SPPARKS).[25] KMCLib is a Python module that employs a backend C++ library. Like the previously mentioned kMC codes, it can handle the simple case where possible events have fixed propensities that do not change over the course of the simulation. However, for more complicated kinds of events, users can provide their own custom Python classes to calculate propensities on the fly. SPPARKS is an open source C++ code that is written in a modular fashion so that it can be readily modified and extended. Typically, modifications come in the form of additional source files that define a class that the SPPARKS documentation calls an "application". These applications are typically very specialized: one of them models a lattice whose sites are either occupied or unoccupied, with no allowance for multiple species of particles at a site; another simulates the diffusion of hydrogen and helium in an erbium lattice; another one implements the Potts mesoscale model of grain growth.[26] Each application depends upon an underlying framework in SPPARKS for handling most of the process for maintaining a set of events and choosing an event to be executed. However, its applications are not entirely free of handling the lower level details of the kMC algorithm. For example, an application is responsible for determining which sites are affected by an event that is executed, and passing those sites to the underlying SPPARKS framework in order to ensure that the set of possible events is properly updated. Also, a given

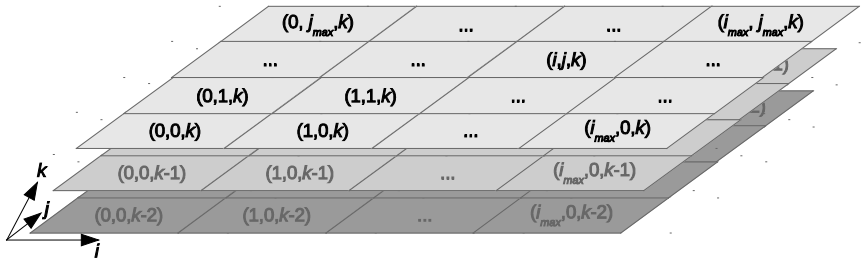
application may use either an on-lattice or off-lattice kMC approach, and this has an impact on how SPPARKS represents a lattice. Since its framework is designed so that its interface is largely the same in both on-lattice and off-lattice applications, a "site" is simply a point in space with an integer label, and a lattice site is simply a site with a static location and a static set of neighboring sites. Currently, there does not appear to be functionality to grow a lattice along a particular direction, so if the simulation domain is set to be a certain size, then that presents a hard ceiling on the thickness of a film that may be grown in a simulation.

This report introduces a new kMC code, "KMCThinFilm", which unlike most of the aforementioned codes, is not an application, but rather a C++ library to be used to create applications that use the kMC method to simulate the growth of thin films. This library is designed to be, on the one hand, more general than codes such as CARLOS,[14] but on the other hand, more tailored for the task of modeling thin films than a very general framework like SPPARKS.[25] It is also designed so that any coding required from users of the "KMCThinFilm" library would pertain to the model that they wish to simulate, rather than to lower level details. That said, certain aspects of its implementation were inspired by the aforementioned kMC codes, especially "kmos"[13,22] and SPPARKS.[25] Section 2 describes the capabilities of the "KMCThinFilm" library and the concepts and some of the algorithms implemented within it. Section 3 describes example applications of the library that showcase its capabilities, and Section 4 shows the results of tests of its parallel algorithms. Finally, Section 5 presents concluding remarks.

2. Concepts and Algorithms of the "KMCThinFilm" Library

A lattice in the "KMCThinFilm" library is modeled as a stack of 2-dimensional arrays of cells, with each cell labeled with a triplet of integer indices, as illustrated in Fig. 1. Periodic boundary conditions always apply to the first 2 indices of a cell, but not the third. In a manner similar to that of SPPARKS,[25] each cell (i, j, k) also effectively contains 2 1-dimensional arrays, one that holds integer values, here named I_{ijk}, and one that holds double precision floating-point values, here named F_{ijk}. The size of these arrays is the same for all cells. The physical meaning of the contents of the arrays is left up to the client application that uses the "KMCThinFilm" library; for a given application, they may be used to identify species of a basis of atoms within a cell, or the spatial coordinates of an atom in a cell in a possibly distorted lattice, or, for a simpler solid-on-solid application, the height of a column of atoms at $(i, j, 0)$. The lattice need not be cubic. If the cells are treated as those of a Bravais lattice with primitive lattice vectors $\mathbf{a}_i$, $\mathbf{a}_j$, and $\mathbf{a}_k$, then the physical location of cell (i, j, k) may be said to be $\mathbf{a}_i i + \mathbf{a}_j j + \mathbf{a}_k k$. The sizes of the arrays in each cell of the lattice, as well as the maximum values of the

first 2 indices of a cell, are fixed when the lattice is initialized. However, additional planes may be added to the lattice at any time step of the simulation, unless the client application explicitly disallows this for the sake of parallel performance. When additional planes are added, by default the elements in the arrays in the cells of each new plane are set to zero. However, this default can be changed. Furthermore, the initial values of I_{ijk} and F_{ijk} in the new cells can be functions of the indices of these cells, or even of the contents of other cells in the lattice. This functionality could be used, for example, to initialize the spatial coordinates of atomic sites in a cell of a possibly distorted lattice.



Cell (i,j,k) contains two arrays:

$$I_{ijk}=\left[i_{ijk}^{(0)},i_{ijk}^{(1)},i_{ijk}^{(2)},\ldots,i_{ijk}^{(N_{\text{Int}})}\right] \qquad F_{ijk}=\left[f_{ijk}^{(0)},f_{ijk}^{(1)},f_{ijk}^{(2)},\ldots,f_{ijk}^{(N_{\text{float}})}\right]$$

Fig. 1 Topmost planes of a lattice whose indices along the *i* and *j* directions are in the interval [0, i_{max}] and [0, j_{max}], respectively. The *i*-, *j*-, and *k*-axes are not necessarily orthogonal.

In the "KMCThinFilm" library, possible events that happen in the lattice are considered to be 1 of 2 kinds, "cell-centered" and "over-lattice". Events of the cell-centered kind originate at or in a neighborhood of a lattice cell, and the propensities of these events are affected by states of cells in that neighborhood. Adsorption, diffusion, and chemical reactions are examples of these kinds of events. Since the propensities of related possible cell-centered events are often calculated using the same or almost the same series of steps, they are treated as groups in the library. To implement a grouping of cell-centered events, the library needs a client application to provide the following:

- A set of "offsets" from the cell about which these events are centered, on which the propensities of the events depend. For example, if the propensities of events at a cell in a square lattice depend upon the states of the nearest neighbors of that cell, then the offsets will include $(-1,0,0)$, $(+1,0,0)$, $(0,-1,0)$, and $(0,+1,0)$, in addition to the always present zero offset, $(0,0,0)$.
- A function or function object[27] that determines the propensities, given an object that encapsulates the above-mentioned offsets and the cell about which an event is centered. The propensities are returned as an array, where

each element is the propensity of a possible event in the group. If an event cannot happen at this cell, its propensity is set to zero.

- A set of functions or function objects, each of which executes a possible event from the group, given the cell about which an event is centered.

Over-lattice events are presumed to have originated from some location well above the lattice, so the propensity of such an event is presumed to be unaffected by the effects of events within the lattice itself. Deposition is the main example of such an event. To implement these, the library needs a client application to provide the following:

- A propensity per lattice plane area, which for deposition would be the number of monolayers deposited per unit time.
- A function or function object that executes the event, given a randomly chosen cell at the top of the lattice.

For both cell-centered and over-lattice events, optional additional sets of offsets may be supplied that indicate the relative locations of cells that would be changed by executing a possible event. These offsets may allow a simulation to run faster in a way that will be described later.

When an event is executed, the indices of the cells of the lattice directly changed by this event are recorded. The indices of cells of any empty planes added by the event are also recorded. The offsets that define cell-centered events are used in conjunction with these recorded indices to determine which possible events have had their propensities changed due to the execution of the event. The set of possible events and their propensities in the simulation is adjusted accordingly. For example, let there be a simulation on a simple cubic lattice containing a single type of cell-centered event that has asymmetrically arranged offsets $(-1,0,0)$, $(+1,0,0)$, and $(0,+1,0)$ (in addition to the always present zero offset) and can change both the cell about which it is centered and the cell to the right of that cell. Figure 2 shows a diagram of the cells changed and affected by this event executing about cell (a_1, b, c). The possible events that are centered near these affected cells will have their propensities changed. Cell (a_1, b, c) is changed, and the cells affected by the change to it have indices that are those of the changed cell plus the "reverse" of the event's offsets, that is, $(a_1 + 1, b, c)$, $(a_1 - 1, b, c)$, and $(a_1, b - 1, c)$. Propensity calculations are performed for each changed or affected cell. The number of calculations performed depends upon whether the aforementioned optional additional sets of offsets have been supplied. If these optional offsets have not been supplied (in addition to the required offsets needed to calculate propensities), then the calculations are performed according to a less efficient "auto-tracking"

algorithm. In this example, 4 propensity calculations would initially be performed, one for the changed cell and one for each of the affected cells. Cell (a_2, b, c), where $a_2 = a_1 + 1$, is also changed, and the cells affected by the change to it are $(a_2 + 1, b, c)$, $(a_2 - 1, b, c)$, and $(a_2, b - 1, c)$. Then, 4 more propensity calculations would be performed, again for the changed cell and the 3 affected cells. Two of these propensity calculations are then redundant, because one of the cells affected by the change to (a_1, b, c), $(a_1 + 1, b, c)$, is also (a_2, b, c), and one of the cells affected by the change to (a_2, b, c), $(a_2 - 1, b, c)$, is also (a_1, b, c). If there are multiple types of cell-centered events (e.g., diffusion to the left, diffusion downward), then the affected cells will be those with indices that are those of each changed cell plus the reverse of the offsets of all of these event types. If N_{ccet} types of cell-centered events contribute N_{off} unique offsets, then the number of propensity calculations for each changed cell will be $N_{ccet} \times N_{off}$. However, if the optional offsets have been supplied, then the propensity calculations are performed according to a so-called "semimanual tracking" algorithm that avoids the redundant propensity calculations. For this example, the optional offsets would be $(0,0,0)$ and $(+1,0,0)$, corresponding to the relative positions of one of the changed cells and the cell to its right. At the beginning of the simulation, from these optional offsets, a new set of offsets is generated that indicates the relative positions of the cells either directly changed or affected by an event, which here would be $(-1,0,0)$, $(0,0,0)$, $(+1,0,0)$, $(+2,0,0)$, $(0,-1,0)$, and $(+1,-1,0)$. Then, when the cell-centered event is executed, a propensity calculation is performed for each cell whose indices are the sum of (a_1, b, c) and one of the new set of offsets.

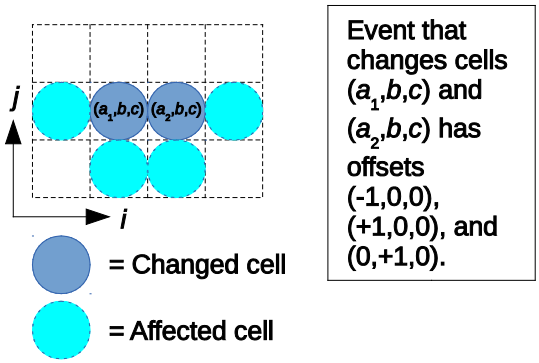


Fig. 2 Diagram of cells in a simple cubic lattice affected by an event with offsets (−1,0,0), (+1,0,0), and (0,+1,0) (in addition to the always present zero offset) that changes the cells (a_1,b,c) and (a_2,b,c)

Since the set of possible events and propensities is incrementally updated rather than generated anew at each time step, propensities are cached and so are not

allowed to be time-dependent. However, there is a way to, say, allow deposition to effectively occur over only a portion of a simulation run, which will be discussed later.

In addition to possible events, so-called "periodic actions" are allowed to occur. Periodic actions may occur either every t units of simulation time or at every m time steps. Like possible events, these actions are defined through functions or function objects. One example of a periodic action would be to dump a "snapshot" of the current state of the lattice to a file. A periodic action is also allowed to change the state of the lattice itself, and there may be cases were this is desirable. For example, in the kMC simulations of quantum dot growth by Meixner et al.,[28] a strain field was updated every 2,500 time steps. If a periodic action changes the lattice, then 1 of 2 things may happen. If the simulation object is given the option to track the indices of the cells changed by a periodic action, then the set of possible events is incrementally updated. Since this option may lead to high-memory consumption if there are many changed cells, the simulation object may be given the option to merely check whether a periodic action has changed the lattice, and then the set of possible events is rebuilt from scratch if a change has taken place.

Choosing a random event at a time step is done via what is called a "solver" in the "KMCThinFilm" library. (This follows similar usage in SPPARKS.[25]) There are currently two solvers to choose from: one implementing the binary tree algorithm by Blue et al.[11] that scales as $O(\log_2 N)$ in the number of possible events N, and one that is a variant of an algorithm proposed by Schulze.[12] In the original algorithm of Schulze, the number of unique nonzero propensity values, n_p, is fixed. An array of length n_p is used to store lists of possible events, where the possible events in the list stored at element m of the array all have propensity p_m. In the variant algorithm used in the library, an associative map is used in place of this array of lists. A key in this map is the propensity of an event and the value associated with this key is the list of possible events with that propensity. Key-value pairs can be added and removed from this map during the course of a simulation, so there is no need to fix the number of unique propensity values. To speed iteration over the map, an auxiliary array[29] is used that contains pointers to the key-value pairs in the map, and iteration is done over this array rather than the map itself. The choosing of events in both the original algorithm and its variant scale should scale linearly with the number of unique propensity values in the system, which is often independent of N.

A simulation itself is considered an object in the "KMCThinFilm" library. It owns a lattice object, which is initialized when the simulation object is initialized. The objects that define possible events and periodic actions (i.e., offsets and various function objects) are then attached to this simulation object. After this, the

simulation is actually run, that is, time stepping begins and events are executed. The objects that define a possible event or periodic action can also be detached from a simulation, and then a simulation can be restarted from where it left off. For example, if one wished to model a system where deposition and diffusion occur together for a period of time T_{dep}, and then, after that period, deposition stops but diffusion continues, one could initialize the simulation, attach the objects defining deposition and diffusion, run the simulation for T_{dep} units of simulation time, detach the function object defining deposition, and then restart the simulation and run it for an additional amount of time.

An abstract interface is provided for random number generators, so client applications can provide their own random number generators and a wrapper class that implements this abstract interface. This may not be necessary since the library does provide a pseudorandom number generator of its own that implements this abstract interface. This generator uses the Mersenne Twister algorithm[30] and is suitable for serial simulations. For parallel applications, the library also provides wrapper classes for two libraries that generate parallel streams of pseudorandom numbers, RngStreams[31,32] and the Dynamic Creator of Mersenne Twister (DCMT) generators.[33,34] Random number generators are not only for use by solvers. A random number generator in the "KMCThinFilm" library is an object in its own right, and it may also be used directly by, for example, function objects that define events.

Parallel kMC simulations can be done using the "KMCThinFilm" library. The library uses an approximate algorithm similar to the one proposed by Shim and Amar[35] and used in SPPARKS.[25,36] In this algorithm, the lattice is split among the parallel processes according to 1 of 2 possible schemes. Figure 3 shows a diagram of an overhead view of a lattice (or of a lattice plane) split across 4 processes according to the "compact" decomposition scheme, which minimizes the length of the boundaries of each partition. In this scheme, there are ghost regions along the edge of each boundary, containing copies of the lattice cells of a neighboring processor. Events are not allowed to originate from cells in these regions, though of course the ghost regions may be affected by events originating from elsewhere. There is also row-based decomposition, where the lattice is split into strips, as shown in Fig 4. In this form of decomposition, if the global in-plane lattice indices are $i \in [0, L_i)$ and $j \in \left[0, L_j\right)$, then the first row has indices $i \in \left[0, L_i/N_{\mathrm{proc}}\right)$ and $j \in \left[0, L_j\right)$, the second has indices $i \in [L_i/N_{\mathrm{proc}}\,, 2\,L_i/N_{\mathrm{proc}})$ and $j \in \left[0, L_j\right)$, and so on, where N_{proc} is the number of processing cores.

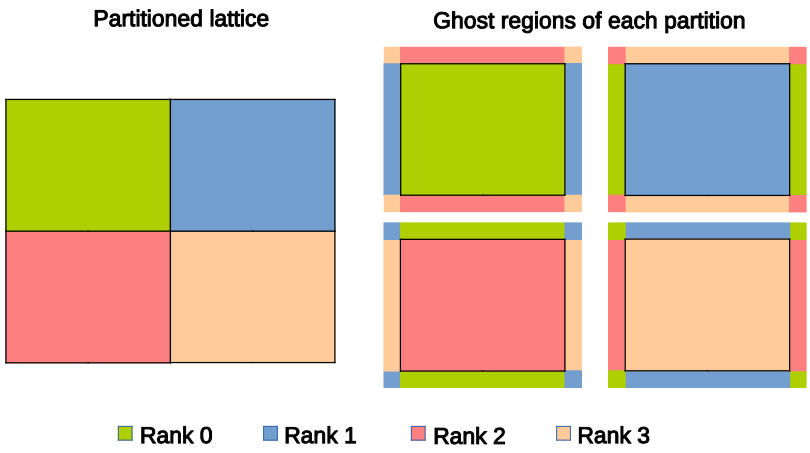


Fig. 3 Lattice partitioned across 4 processes according to the compact scheme. Each process is assigned a rank from 0 to 3. The ghost regions along the boundaries of each partition are copies of the lattice cells of a neighboring processor, and in the diagram, the processor from which they are copied is shown via their color. Periodic boundary conditions apply.

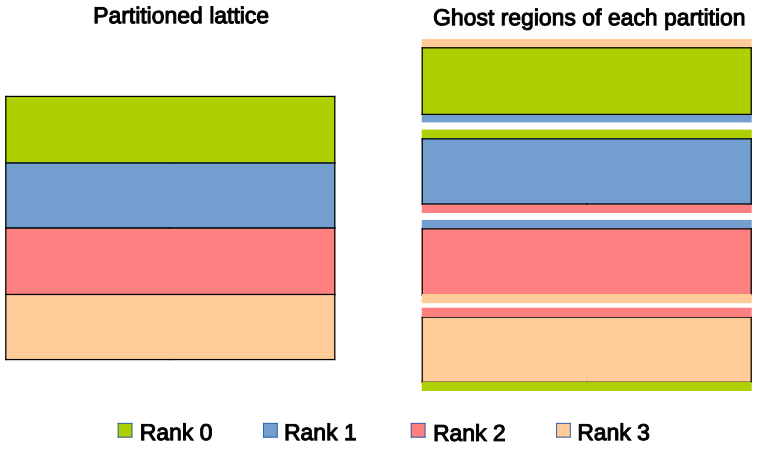


Fig. 4 Lattice partitioned across 4 processes according to the row-based scheme. The ghost regions along the boundaries of each partition are copies of the lattice cells of a neighboring processor, and in the diagram, the processor from which they are copied is shown via their color. Periodic boundary conditions apply.

Attempting to do kMC simulations on each partition of the lattice would lead to problems at the partition boundaries, since the events done on each partition could lead to conflicting effects on the ghost cells. To avoid this, an approximation is made where each partition is further subdivided into "sectors", as illustrated in Fig. 5 for compact decomposition. In compact decomposition, each partition is subdivided into 4 quadrants, which become the sectors. In row-based decomposition, there would be 2 sectors per partition. In the figure, the active sectors happen to be the upper left quadrants of the lattice partitions. At any given time in the simulation, they could be the lower left, lower right, or upper right quadrants, so long as the relative locations of these active sectors are the same for

all processors, that is, “all” upper left, “all” lower left, and so on. An active sector is one where events are allowed to execute. While this imposes an unphysical restriction on where events can execute—hence why this algorithm is approximate—it also prevents parallel processes from executing conflicting events at the boundaries. Because the active sectors all have the same relative location, the regions that are affected by events happening within them do not overlap.

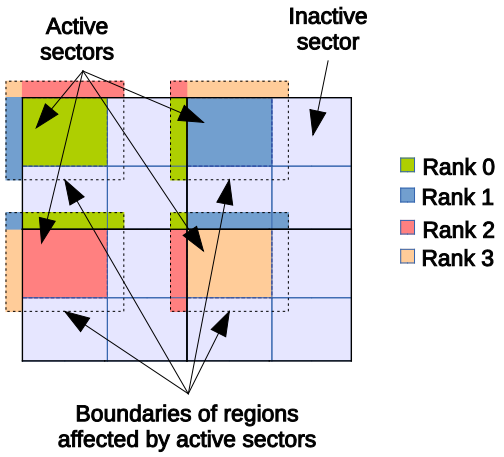


Fig. 5 Sector boundaries for a lattice decomposed across 4 processes according to the compact scheme. Partition boundaries are indicated by thick solid lines, while the sector boundaries are indicated by thinner solid lines. Active sectors are shown in the color corresponding to the rank of the partition to which they belong. The dotted lines show the boundaries of the regions affected by events that occur within the active sectors. The parts of these regions that affect ghost cells are shown in the color corresponding to the ranks of the cells of which the ghost cells are copies. Here, the active sector happens to be the upper left of each partition, but at a given time, the active sector could be in the upper right, lower right, or lower left.

With the sectors now defined, the approximate kMC algorithm can proceed on each processor according to the following algorithm. First, the global time t on all processors is initialized to zero, while the variable t_{stop} is initialized to some nonzero value. Until t exceeds the desired global simulation time t_{max}, the sectors are iterated over in a loop. For each sector visited, the local time t_{local} is initialized to zero. The number of lattice planes at each partition is adjusted so that it is the same for all partitions, unless the client application has explicitly disallowed this in order to avoid the parallel communication that would be needed for this step. The ghost cells are then updated, and the set of possible events is updated accordingly. A normal serial kMC algorithm is then run on the cells within the sector, with t_{local} incremented as each event is executed until $t_{local} > t_{stop}$. However, the event that would cause t_{local} to exceed t_{stop} is not executed. If events have caused additional lattice planes to be added, then the number of lattice planes at each partition is again adjusted to ensure that it is the same for all partitions. The off-processor cells that correspond to the ghost cells, which may have changed due to the events that have

occurred, are then updated, and again, the set of possible events affected by changes to the ghost cells is updated. After the iteration, the global time t is incremented by t_{stop}, and the value of t_{stop} is updated if needed. The updates of the ghost cells that are performed when a sector is visited do not involve communicating the entirety of the ghost cell regions bordering a sector, only the communication of changes to each region. Currently, this communication is done via Message-Passing Interface (MPI).[37]

The value of t_{stop} may be determined by various time-stepping schemes. The simplest of these is to set t_{stop} to a fixed value. The other schemes are various kinds of adaptive algorithms, which attempt to determine a reasonable value of t_{stop} from the propensities of the possible events in the simulation. In all of these algorithms, the time step has the general form,

$$t_{stop} = \frac{n_{stop}}{F_{stop}}, \tag{1}$$

where n_{stop} is an adjustable parameter, and F_{stop} is a function that determines the particular adaptive time-step scheme. There are 2 choices currently available for F_{stop} in the "KMCThinFilm" library:

1) **Maximum Single Propensity.** This is the maximum propensity of all currently possible cell-centered events in the simulation. This is a simplified version of the adaptive method of determining F_{stop} recommended by Shim and Amar.[35]

2) **Maximum Average Propensity.** This is the maximum of the average propensities per possible cell-centered event from each sector. Here, $F_{stop} = \max\left(p_s^1, p_s^2, \ldots, p_s^{N_{\mathrm{proc}}}\right)$, where for the case of compact decomposition, $p_s^n = \max\left(p_{s,UL}^n, p_{s,LL}^n, p_{s,LR}^n, p_{s,UR}^n\right)$ and $p_{s,UL}^n$ is the average propensity of the events in the upper-left sector of partition n, that is, the sum of the propensities of all possible events in that sector divided by the number of those possible events, while similarly, $p_{s,LL}^n$, $p_{s,LR}^n$, and $p_{s,UR}^n$ are the mean propensities in the lower left, lower right, and upper right sectors of partition n. This approach was recommended by S. Plimpton et al.[36]

Over-lattice events are excluded from the adaptive time-step calculations. The propensity of an over-lattice event depends upon the in-plane dimensions of the lattice, while the propensities of cell-centered events do not. Because of this, mixing over-lattice and cell-centered events biases adaptive time-step calculations so that they tend to lead to unreasonably small time steps for larger lattices. If there are no possible cell-centered events at the beginning or restart of a simulation that can be

used to determine F_{stop}, then at each sector, an event is executed (e.g., deposition) that should cause one or more cell-centered events to be possible. The simulation time is then incremented by the time taken to execute the slowest event among all of the partitions.

The proper value of n_{stop} is a trade-off between speed and accuracy, with larger values leading to fewer updates of ghost cells. Some kinds of simulations can tolerate higher values of n_{stop}. For example, in a parallel island coarsening simulation where time stepping was done according to the maximum propensity algorithm of Shim and Amar, n_{stop} could be as high as 10.0 without significantly degrading accuracy.[38] If n_{stop} is too large, a so-called "shish kebob" artifact may be seen, where islands, grains, or other simulated features tend to congregate near sector boundaries.[36]

Running a parallel simulation changes how periodic actions are run. In a serial simulation, if a periodic action executes, it executes shortly after an event has been executed. In a parallel simulation, if a periodic action executes, it executes shortly after t_{stop} has been incremented, that is, outside of the looping-over sectors. This allows periodic actions to use MPI calls for parallel communication, since a periodic action will execute the same number of times on every processor.

3 Example Applications

These examples of kMC models were chosen because they have features that generally cannot be specified via an input file that would be supplied to most of the available "off-the-shelf" kMC applications such as "CARLOS,"[14,15] "CHIMP,"[16,17] "MONTY,"[18,19] or Graph Theoretical kMC.[20,21] Rather, they would tend to require at least some ad hoc coding for their implementation. One of these examples is a pair of relatively simple qualitative models of the formation of islands on a patterned substrate that tends to draw particles toward sites of a regular superlattice grid. The other is of a pseudo-polycrystal formed by ballistic deposition, where there is a single computational lattice and grains are identified via an integer label at a lattice cell rather than an actual grain orientation. These examples reproduce trends in the literature and show some of the possibilities offered by the "KMCThinFilm" library.

3.1 Patterned Substrate

The two qualitative models of patterned substrates presented here were developed by Kuronen et al.[5] The lattice in these models is simple cubic, and it is assumed that there are no overhangs or internal vacancies. In both models, the propensity for

diffusion by hopping of a particle at lattice cell (i, j, k) to a vacant cell with in-plane indices $(i \pm 1, j)$ or $(i, j \pm 1)$ may be written as

$$p_{hop} = \frac{k_B T}{h} \exp\left(-\frac{E(i,j)}{k_B T}\right), \tag{2}$$

where k_B is Boltzmann's constant, h is Planck's constant, T is the temperature, and $E(i, j)$ is a model-dependent energy barrier whose value depends upon the in-plane cell indices i and j. In what Kuronen et al.[5] describe as "model A," this barrier takes the form

$$E(i, j) = E_s(i, j) + nE_n\,, \tag{3}$$

where n is the number of lateral nearest neighbors, E_n (= 0.18 eV) is a parameter indicating the strength of the bond to a lateral nearest neighbor, and $E_s(i, j)$ is a periodically varying energy barrier. The pattern of periodicity is shown in Fig. 6. The unit cell, or domain, that forms this pattern has dimensions of 22×22 lattice cells, and it is tiled in a 16×16 arrangement. Within a domain, E_s varies linearly from 0.65 eV at the edge of the domain to 0.85 eV at its center.

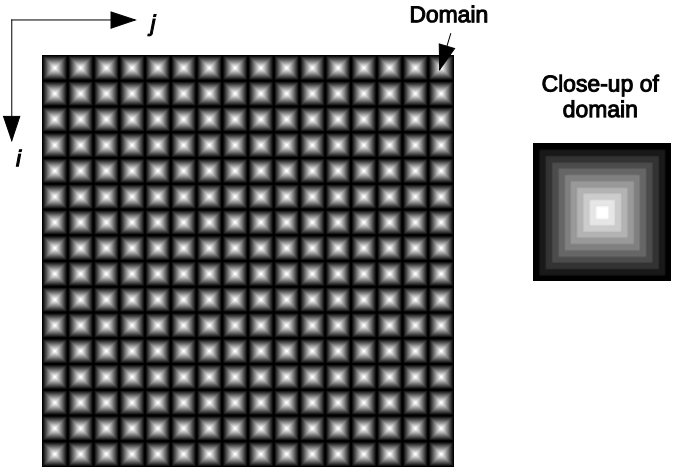


Fig. 6 Variation of E_s and E_d in patterned substrate models A and B from Kuronen et al.[5] Here $i,j \in [0,352)$ and the size of an individual domain is 22×22. For E_s, brighter colors constitute higher values, while for E_d, the reverse is true.

In what Kuronen et al.[5] describe as "model B," the barrier takes a different form,

$$E(i, j; D) = E_s + nE_n + H(i, j; D)E_d(i, j). \tag{4}$$

Here, n and E_n are as before. E_s is now a constant value of 0.75 eV. $E_d(i, j)$ has the pattern of variation shown in Fig. 6, but varies linearly from 0.02 eV at the edge of a domain to zero at its center. The variable D indicates the direction of a particle hop. If a particle moves toward the edge of a domain and is within half the lateral size of a domain from that edge, then $H(i, j; D) = 1$; otherwise, it is zero.

In a later paper[39] by Kuronen et al.,[40] a particle initially randomly deposited at a lattice cell with in-plane lattice indices (i, j) will seek out cells with in-plane indices $(i \pm 1, j)$, $(i, j \pm 1)$, and $(i \pm 1, j \pm 1)$ that the most lateral nearest neighbors. If such a cell is found, the particle will then move to it. This model of deposition will be used here. The flux for deposition is taken to be $F = 0.0033$ ML/s, and deposition continues until a coverage of 15% has been reached.

Conceptually the lattice is a 3-dimensional cubic lattice, but since there are no vacancies or overhangs, the simulation can be done with the "KMCThinFilm" library using a 2-dimensional computational lattice, that is, a lattice with a single plane, for computation. The array of integers I_{ij0} in computational lattice cell $(i, j, 0)$ is set to a size of one and is used to store the height of the column of particles at cell $(i, j, 0)$ of the "true" 3-dimensional lattice being modeled. The following function objects are needed for simulation with the "KMCThinFilm" library:

- A function object to execute a deposition event. When a deposited particle seeks out nearby cells with the most nearest neighbors, there may be more than 1 cell with the most nearest neighbors, so a random number will be used to break any ties. Accordingly, this function object contains a pointer to the random number generator used in the simulation.
- Four function objects to execute a hop in each of the 4 possible directions. These objects can easily be instantiations of a common function object class whose constructor takes an argument that determines the hopping direction, and that is what has been done here.
- A function object for determining the propensities for hopping in each of the 4 directions. For model A, this function object contains a pointer to a 22×22 array containing the values of $E_s(i, j)$ in a domain. For model B, the object contains a pointer to an array with the same dimensions but containing the values of $E_d(i, j)$ in a domain.

The same 5 offsets are needed for the calculation of each hopping propensity: $(0,0,0)$, $(0,+1,0)$, $(0,-1,0)$, $(+1,0,0)$, and $(-1,0,0)$. To obtain a coverage of $\Theta_c = 15\%$, the simulation ran for Θ_c/F units of simulation time. Afterwards, the deposition event was removed from the simulation, which then ran for an additional $0.1\,\Theta_c/F$ units of simulation time. A function object implementing a periodic action was used to print a snapshot of the lattice to a file every $0.05\,\Theta_c/F$ units of simulation time. The Mersenne Twister algorithm with a period of $2^{19937} - 1$[30] was used to generate pseudorandom numbers.

In the later paper by Kuronen et al.,[40] it was shown that similar trends were seen for both models A and B. At relatively low temperatures, small groups of islands or

islands with very irregular boundaries tended to form near the centers of the domains. As the temperature rises, the islands tend to become more compact and arrange themselves more clearly into a square array, and with further increase in temperature, the islands tend to grow larger and vacancies in the array of islands appear more frequently. This can be seen in Figs. 7 and 8, which show some results of simulations of these models using a client application of the "KMCThinFilm" library. Average wall clock times where taken on these simulations on a 2.3-GHz Intel Xeon E5-2698 v3 node of a computational cluster and the average time per event (i.e., the wall clock time divided by the total number of events executed in a simulation) are shown in Tables 1 and 2. (While the simulations are serial, using a cluster allows several simulations to be performed simultaneously.) For each model type, temperature, and solver, the averages of the wall clock times and event counts were taken over 100 simulation runs. The average time per event ranges from about 2 to 4 μs. This time per event tends to be largest for the smallest temperatures, likely because as the temperature decreases, the number of hopping events decreases while the number of deposition events—which happen to be more expensive to execute than hopping in these particular models—remains approximately the same. As expected, both the wall clock time and the time per event tend to be smaller when semimanual tracking is used, though given the short running times of these simulations, its benefits are marginal here.

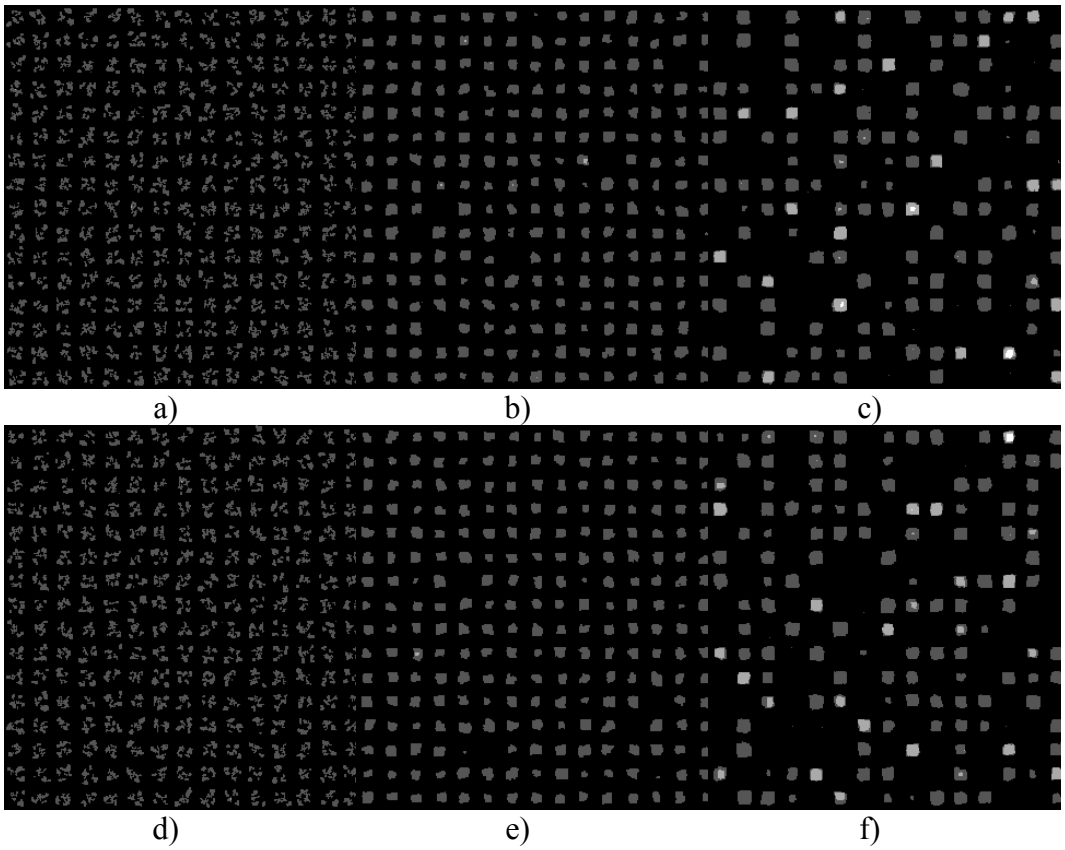


Fig. 7 Arrangement of islands at the end of the simulation of model A for temperatures a) T = 340 K, b) T = 390 K, and c) T = 440 K, using the solver based on the Schulze[12] algorithm, and for temperatures d) T = 340 K, e) T = 390 K, and f) T = 440 K, using the binary tree algorithm. The grey level at a point indicates the height of the column of particles there.

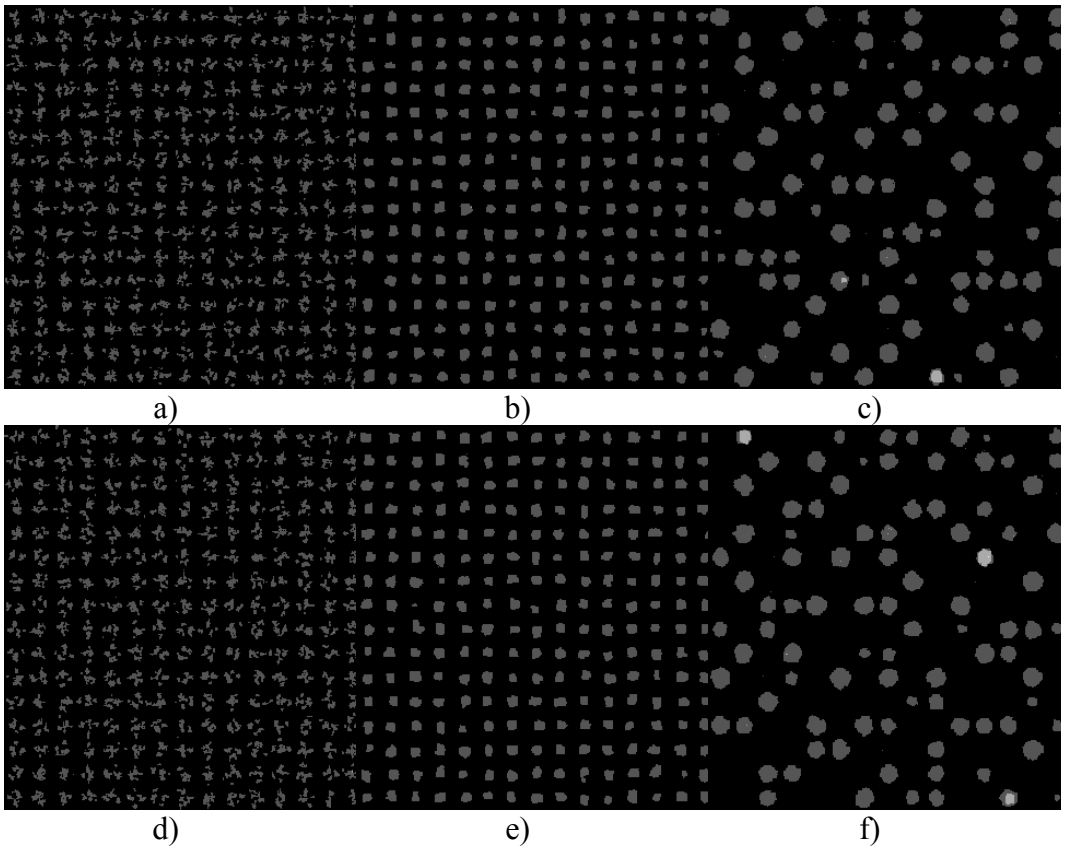


Fig. 8 Arrangement of islands at the end of the simulation of model B for temperatures a) T = 340 K, b) T = 390 K, and c) T = 440 K, using the solver based on the Schulze[12] algorithm, and for temperatures d) T = 340 K, e) T = 390 K, and f) T = 440 K, using the binary tree algorithm. The grey level at a point indicates the height of the column of particles there.

Table 1 Average wall clock times and average times per event for simulations of model A on an Intel Xeon 2.3-GHz node of a computational cluster

Solver	Temperature (K)	Auto-tracking		Semimanual tracking	
		Avg. wall time (s)	Avg. time per event (μs)	Avg. wall time (s)	Avg. time per event (μs)
Schulze algorithm	340	5.8	2.7	5.5	2.6
	390	14.5	1.9	13.7	1.8
	440	133.1	1.6	122.2	1.5
Binary tree	340	7.8	3.7	7.5	3.5
	390	20.4	2.6	19.1	2.5
	440	195.8	2.4	183.5	2.2

Table 2 Average wall clock times and average times per event for simulations of model B an Intel Xeon 2.3-GHz node of a computational cluster.

Solver	Temperature (K)	Auto-tracking		Semimanual tracking	
		Avg. wall time (s)	Avg. time per event (μs)	Avg. wall time (s)	Avg. time per event (μs)
Schulze algorithm	340	4.2	4.8	4.0	4.7
	390	11.3	2.2	10.8	2.1
	440	170.9	1.6	159.2	1.5
Binary tree	340	4.9	5.7	4.9	5.7
	390	15.2	2.9	14.9	2.9
	440	261.5	2.5	251.3	2.4

3.2 Pseudo-polycrystal Formed by Ballistic Deposition

Here, the lattice is face-centered cubic (fcc) with the surface normal along the (111)-direction, and a cell with indices (i,j,k), $i,j \in [0,100)$, is presumed to contain a particle with a physical location of $\mathbf{a}_i i + \mathbf{a}_j j + \mathbf{a}_k k$, where primitive lattice vectors $\mathbf{a}_i$, $\mathbf{a}_j$, and $\mathbf{a}_k$, are oriented as follows in respect to the unit vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ pointing along the Cartesian x-, y-, and z-axes.

$$\begin{aligned} \mathbf{a}_i &= \frac{a}{2\sqrt{2}}\left(\sqrt{3}\hat{\mathbf{x}} + \hat{\mathbf{y}}\right) \\ \mathbf{a}_j &= \frac{a}{\sqrt{2}}\hat{\mathbf{y}} \\ \mathbf{a}_k &= a\left(\frac{1}{4}\sqrt{\frac{2}{3}}\hat{\mathbf{x}} + \frac{\sqrt{2}}{4}\hat{\mathbf{y}} + \frac{1}{\sqrt{3}}\hat{\mathbf{z}}\right). \end{aligned} \tag{5}$$

The lattice constant is a, and $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are parallel to the lattice planes. Since no particular material is being modeled here, a is simply taken to be equal to 1.

The pseudo-polycrystal model presented here is similar to that in work by Wang and Clancy,[6] though while that work employed both ballistic deposition and deposition where a particle can bounce off of surfaces to which it did not stick, here deposition is always ballistic. A depositing particle appears above the lattice, travels with an incidence angle θ in respect to the surface normal (and an azimuth that here is set to zero), sticks to the first available cell within a distance $a/\sqrt{2}$ from the line of incidence, and if this cell does not have at least 3 nearest neighboring particles, the depositing particle migrates to a nearby vacant cell that does. Conceptually, a lattice cell (i,j,k) with $k < 0$ is considered part of the substrate, so any cell in the first plane of the computational lattice (i.e., $k = 0$) is in contact with the substrate and thus treated as having 3 nearest neighboring particles. Afterwards, the deposited particle acquires an integer label that identifies it as belonging to a grain.

If possible, this integer grain label is randomly chosen to be one of the grain labels of its nearest neighbors. Otherwise, it is set to an integer value that has not yet been used. A particle may diffuse by hopping to a vacant nearest-neighboring cell provided, 1) that both the particle and the vacant cell have at least 3 other particles as nearest neighbors, and 2) that all of the nearest neighboring particles of the vacant site have the same grain label as the particle attempting to diffuse. If these conditions are satisfied, then the propensity for hopping of a particle is taken to be

$$p_{hop} = \nu_0 \exp\left(-\frac{E_d + E_n(n_{particle} - \alpha n_{vacant})}{k_B T}\right), \tag{6}$$

where $n_{particle}$ is the number of nearest neighbors of the particle, n_{vacant} is the number of nearest neighbors of the vacant cell, E_d is the base diffusion barrier to hopping, E_n and α are parameters affecting the influence of neighbors on the hopping of a particle, k_B is Boltzmann's constant, and T is the temperature. Following Voter,[1] the attempt frequency ν_0 is set to 10^{13} Hz. The chosen values for E_d, E_n, and α have been set mostly arbitrarily to 1.0 eV, 0.25 eV, and 0.3, respectively. The temperature was set to 700 K. The simulation continues until a coverage $\Theta_c = 20$ monolayers (ML) has been approximately reached, that is, it continues until the simulation time reaches Θ_c/F, where F is the deposition flux. Simulations were done for $F = 0.1$ ML/s and $F = 10$ ML/s.

The array of integers I_{ijk} in computational lattice cell (i, j, k) is set to a size of one and is used to store the grain label, and a grain label value of zero indicates that the cell is unoccupied. The Mersenne Twister algorithm[30] with a period of $2^{19937} - 1$ was used to generate pseudorandom numbers, and the binary tree solver was used to randomly select events. The following function objects were used for simulation with the "KMCThinFilm" library:

- A function object for calculating the hopping propensity. This function object is associated with a total of 157 offsets: offset (0,0,0), which is always present; the offsets indicating the locations of nearest neighboring cells of the particle that may hop; and the offsets indicating the locations of the twelve nearest neighboring cells of each of the 12 sites to which the particle may hop. Of these 157 offsets, only 55 are unique, but this only affects memory consumption, not execution time.
- Twelve function objects for executing a hopping event. Again, these were all instances of the same class.
- A function object executing deposition. This object contains *all* of the logic used for ballistic deposition. This logic, by design, is *not* part of the "KMCThinFilm" library itself.

- A function object implementing a periodic action to print a snapshot of the lattice to a file. A snapshot was printed every $0.05\ \Theta_c/F$ units of simulation time.

Qualitative relationships between the angle of incidence for deposition and growth morphology can be seen in Figs. 9–11, which show slices of the deposited thin film for a deposition flux of $0.1\ \mathrm{ML/s}$ and angles of incidence varying from 0° to 85°. As the angle of incidence increases, the growth angles of the grains in respect to the surface normal increase as well. Also, there is a transition from columnar growth, where the grains are in contact with one another, to dendritic growth, where grains grow in needle-like structures. Both of these trends are physically reasonable and consistent with previous kMC simulations.[6,41]

The relationship between the growth angle of grains and the angle of incidence for deposition in the current simulations is also apparent from the histograms and other data shown in Figs 12 and 13. The histograms show the distribution of the growth angle and "reversed" azimuth (that is, angle in respect to the negative x-axis) of the grains grown with a deposition flux of 10 ML/s. The growth angle and azimuth for a grain are taken to be the angles with respect to the z-axis and negative x-axis of the line that minimizes the sum of the squares of the distances[42] between itself and the particles in that grain. For a given angle of incidence, such lines were determined for grains from 10 different simulations, but least-squares lines were not determined for grains with less than 50 particles. For $\theta = 0°$, the grains tend to grow at a slight tilt from the surface normal rather than straight up, but there is no particular vertical plane with which the grains tend to align. As the angle of incidence increases, the growth angle tends to increase, and the grains also tend to be increasingly aligned with the planes that contain the deposition trajectories, that is, those parallel to the xz-plane. Once the grains are more consistently aligned parallel to the xz-plane (i.e., when their growth angles are about 45° or more), their growth angles can be meaningfully compared with the 2-dimensional theory of Tait et al.[43] on the relationship between growth angle β of columnar grains and the incidence angle of deposition, which is as follows.

$$\beta = \theta - \sin^{-1}\left(\frac{1-\cos\theta}{2}\right). \tag{7}$$

The data in Fig. 13 show that the growth angles in the simulated grains are less than those predicted by Tait et al.[43] This is consistent with results for ballistic deposition in the work of Wang and Clancy.[6]

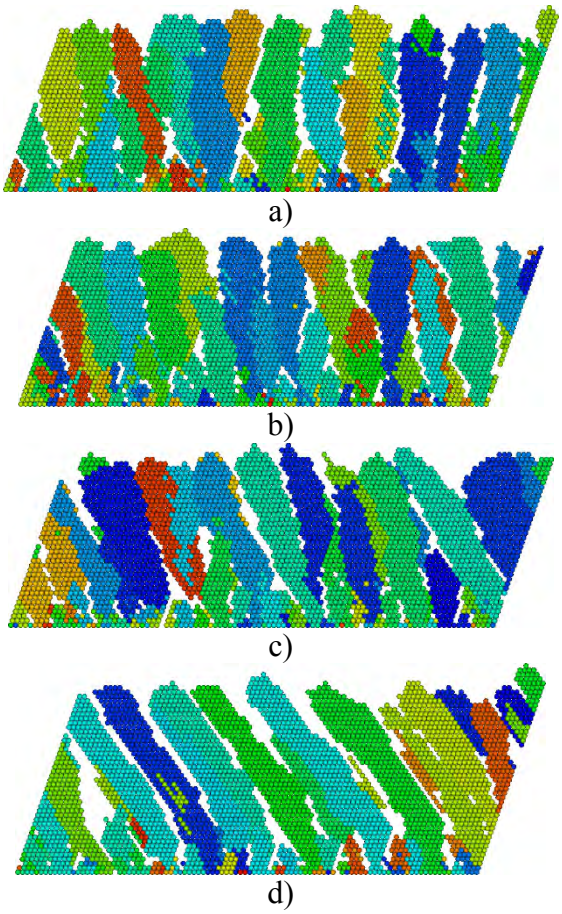
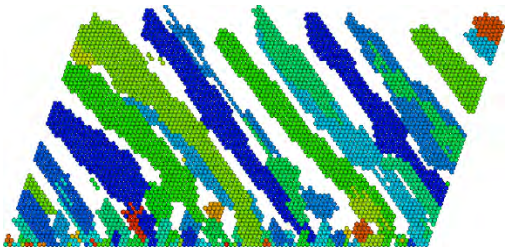
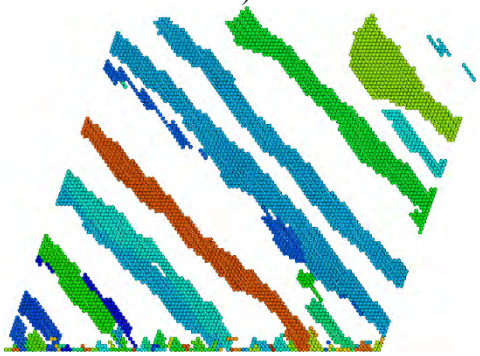


Fig. 9 Slice of simulated thin film for flux $F = 0.1$ ML/s and angles of incidence from a) $\theta = 0°$, b) $\theta = 15°$, c) $\theta = 30°$, and d) $\theta = 45°$. Here, the x-axis (not shown) points to the right, and the y-axis (also not shown) points out of the page.



a)



b)

Fig. 10 Slice of simulated thin film for flux F = 0.1 ML/s and angles of incidence from a) θ = 60° and b) θ = 75°. Here, the x-axis (not shown) points to the right, and the y-axis (also not shown) points out of the page.

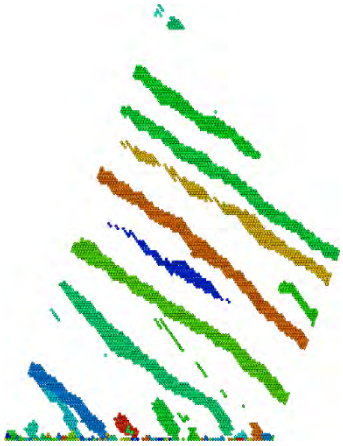
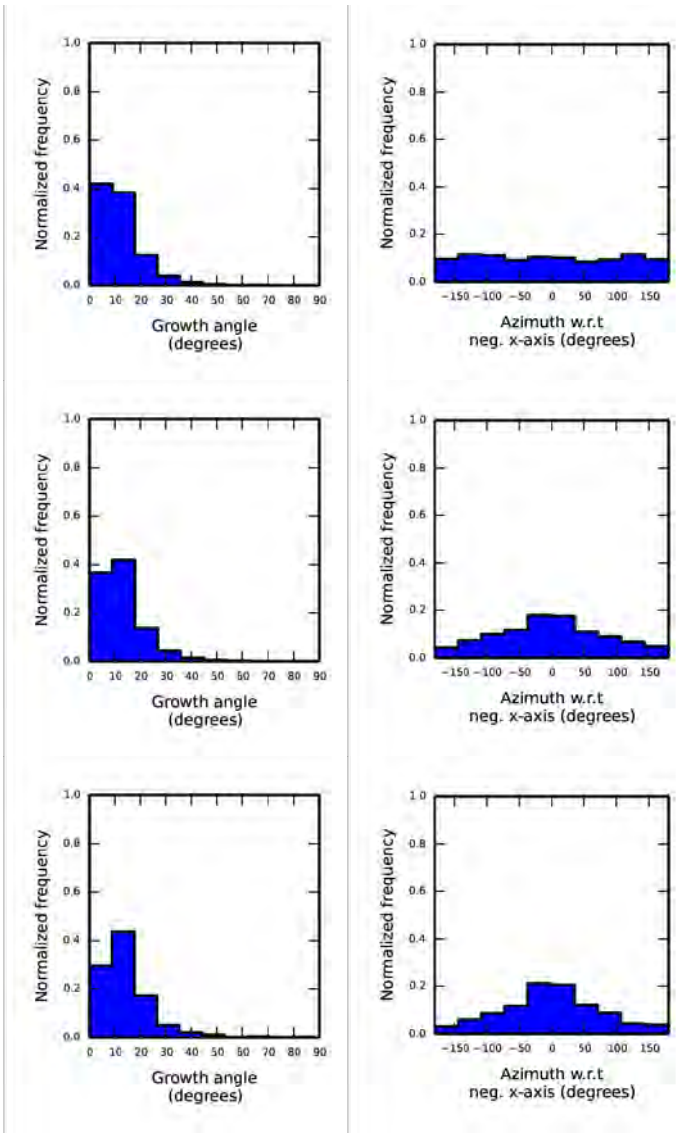


Fig. 11 Slice of simulated thin film for flux F = 0.1 ML/s and angle of incidence θ = 85°. Here, the x-axis (not shown) points to the right, and the y-axis (also not shown) points out of the page.

a)



Incidence angle
0°
Growth angle
Mean: 12.8°
Std. dev.: 9.8°
Azimuth
Mean: –2.85°
Std. dev.: 104.4°

b)

Incidence angle
15°
Growth angle
Mean: 13.4°
Std. dev.: 10.0°
Azimuth
Mean: –1.35°
Std. dev.: 82.5°

c)

Incidence angle
30°
Growth angle
Mean: 14.7°
Std. dev.: 10.4°
Azimuth
Mean: –1.50°
Std. dev.: 74.7°

Fig. 12 Histograms of the growth angle and azimuth (with respect to the negative *x*-axis) of grains of films grown with a deposition flux of 10 ML/s and angles of incidence varying from 0° to 30°.

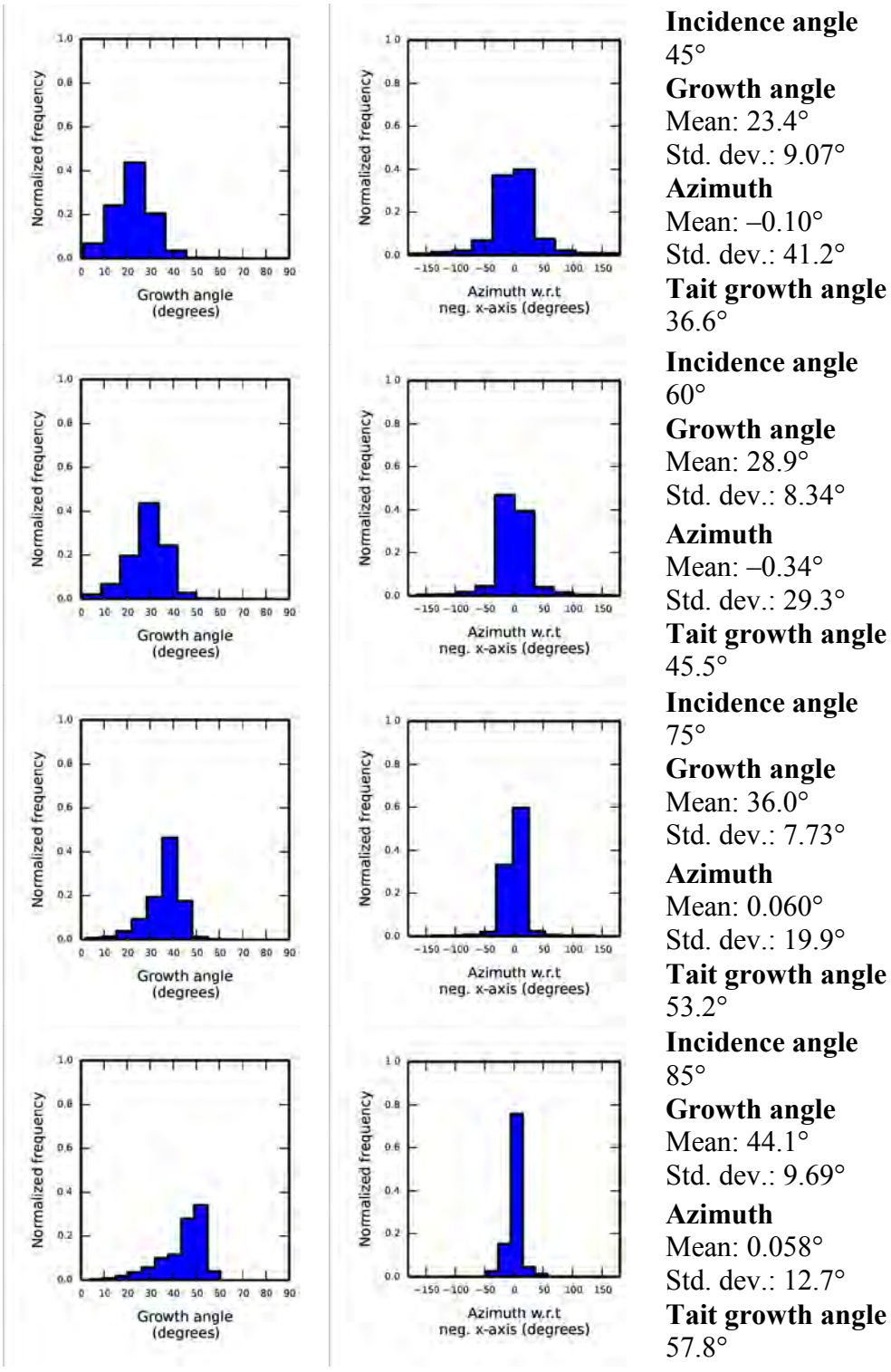


Fig. 13 Histograms of the growth angle and azimuth (in respect to the negative *x*-axis) of grains of films grown with a deposition flux of 10 ML/s and angles of incidence varying from 45° to 85°. The mean and standard deviations of the growth angle and azimuth, as well as the growth angle predicted from the theory of Tait et al.[43] are also shown.

In the current simulations, the angle of incidence also affects the density of the film. Here, the relative density is taken to be the number of particles in the deposited film divided by the number of particles that would be in a fully occupied lattice that is the same height as the film. (This implies that internal vacancies in the film are treated as closed pores, while valleys in the surface of the film are treated as open pores.) Figure 14 shows the relationship between the angle of incidence and relative density for 2 different deposition fluxes, 0.1 and 10 ML/s. For a given angle of incidence, the relative density is averaged from the results of 10 different simulations. Again, the qualitative trends are similar to those seen in previous simulations;[6,41,43] initially the density is largely independent of the angle of incidence, but after growth transitions from columnar to dendritic, the increasing gaps between the grains contribute to a steady decrease in the density of the film. The film grown at a deposition flux of 0.1 ML/s is denser than one grown at 10 ML/s because the simulation continues until a certain amount of coverage has been reached, so a smaller deposition flux corresponds to a larger simulation time and more opportunities for diffusing particles to fill in vacancies.

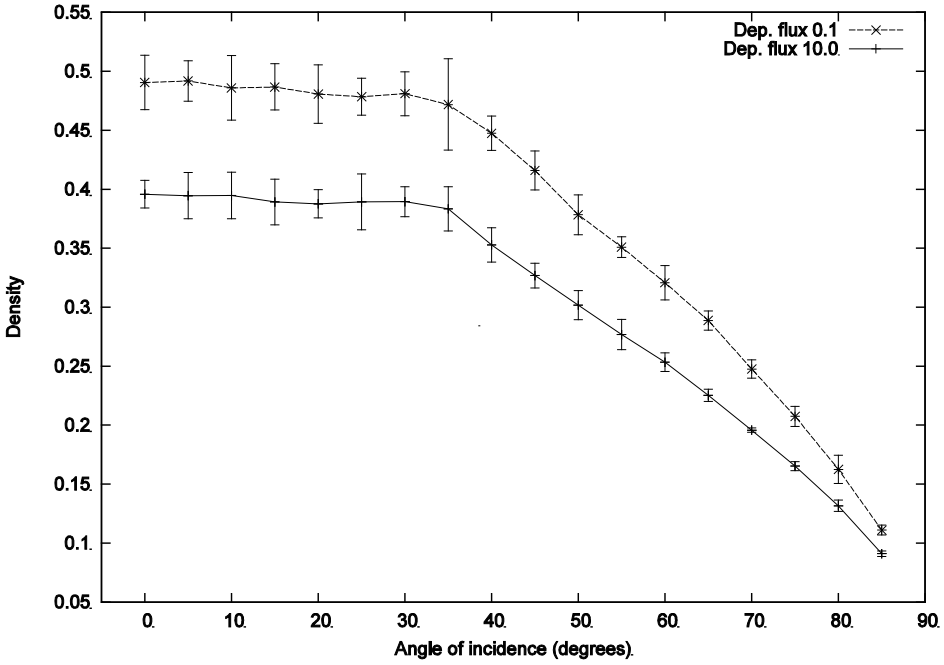


Fig. 14 Relative density of a simulated film deposited by ballistic deposition versus the angle of incidence. A film with a relative density of 1.0 has no vacancies. Error bars represent a 99.7% confidence interval.

The average wall clock times taken by these simulations on a 2.3-GHz Intel Xeon E5-2698 v3 node of a computational cluster and the average times per event, that is, the wall clock time divided by the total number of events executed in a simulation, are shown in Tables 3 and 4. (While the simulations are serial, using a

cluster allows several simulations to be performed simultaneously.) The averages of the wall clock times and event counts were taken over 10 simulation runs. For the simulations with the high-deposition flux, the wall clock times range from about 1 to 2 min. The simulations with the lower deposition flux generally have executed roughly a hundred times as many events, and the wall clock times are accordingly larger, ranging from about 0.8 to 2.6 h. The time per event for these simulations is about an order of magnitude longer than that of the simulations in Section 3.1, and there are at least 2 issues that cause these simulations to take significantly longer. First, the possible events in each simulation contribute a total of 55 unique offsets, as opposed to 5 for the patterned substrate: the offset $(0,0,0)$ and the offsets associated with the nearest neighbors. Second, the number of types of cell-centered events for the pseudo-polycrystal, 12, is 3 times larger than it is for the patterned substrate. These 2 considerations alone mean that when an event is executed in the pseudo-polycrystal simulation and auto-tracking is used, there are 660 (i.e., 55×12) propensities calculated for each cell changed by an event, while only 20 (i.e., 5×4) propensity calculations are done for each cell changed by an event in the patterned substrate simulation. Use of semimanual tracking does cut down on the simulation times by about a factor of 1.3 to 1.4.

Table 3 Wall clock time and average times per event spent in simulations with deposition flux F = 10 ML/s.

Angle of incidence (°)	Auto-tracking		Semimanual tracking	
	Average wall time spent in simulation (s)	Average wall time per event (μs)	Average wall time spent in simulation (s)	Average wall time per event (μs)
0	80.9	71.8	62.2	55.5
15	79.2	71.4	60.9	54.8
30	73.5	71.9	59.2	58.2
45	93.2	75.3	72.6	58.9
60	103.6	73.8	80.1	56.1
75	120.9	77.4	95.5	60.9
85	125.1	76.9	96.5	59.5

Table 4 Wall clock time and average times per event spent in simulations with deposition flux F = 0.1 ML/s.

Angle of incidence (°)	Auto-tracking		Semimanual tracking	
	Average wall time spent in simulation (s)	Average wall time per event (μs)	Average wall time spent in simulation (s)	Average wall time per event (μs)
0	9,233.8	78.3	6,413.6	53.8
15	8,579.7	72.3	6,545.3	55.9
30	8,964.2	78.8	6,402.1	56.7
45	8,837.8	72.7	6,714.2	55.8
60	8,595.7	75.1	6,276.5	55.0
75	5,871.0	72.9	4,410.2	52.3
85	3,980.0	65.9	2,759.8	45.4

4. Parallel Scalability

Kratzer[44] has claimed that ". . . the scalability of parallel kMC simulations for typical tasks is practically limited to four or eight parallel processors with the currently available parallel algorithms," and the results here suggest that this may be at least partially true. Parallel scalability does tend to decay rapidly as the number of processing cores increases. On the other hand, this tendency decreases as the size of the lattice used in the simulations increases.

Tests of parallel scalability have been done for a client application of the "KMCThinFilm" library that implements a solid-on-solid island coarsening model from Shi, Shim, and Amar.[38] In this model, the lattice is simple cubic, and a particle in a cell of this lattice can diffuse with propensity

$$D_m = \nu_0 \exp\left(-\frac{E_0 + nE_b}{k_B T}\right), \tag{8}$$

where attempt frequency $\nu_0 = 10^{12}$ s^{-1}, $E_0 = 0.4$ eV, n is the number of nearest in-plane neighbors, $E_b = 0.1$ eV, k_B is the Boltzmann constant, and temperature $T = 250$ K. If a particle diffuses to a cell that is above one or more empty cells, it falls until it lands atop an occupied cell. The "falling" here is only a conceptual part of the model; computationally, the particles in the cubic lattice are represented using a 2-dimensional computational lattice where, as in the solid-on-solid model of the patterned substrate in Section 3.1, the array of integers I_{ij0} in computational lattice cell $(i, j, 0)$ is set to a size of 1 and used to store the height of the column of particles at cell $(i, j, 0)$ of the "true" lattice being modeled. The deposition flux F was set to 1 ML/s, and deposition continued until coverage of about 10% was reached. Every 50 s of simulation time, each MPI process printed a snapshot of the part of the

lattice that it owned. The total simulation time was 10^3 s. Parallel streams of random numbers are provided via the RngStreams library.[31]

For strong scalability, parallel speed-up is defined as

$$S(N_{\text{proc}}) = \frac{t_1}{t_{N_{\text{proc}}}}, \tag{9}$$

where t_m is the wall clock time taken to complete a simulation over a domain distributed over m processing cores. For consistency, the approximate sectoring algorithm is used to determine t_1. In the ideal case, $S(N_{\text{proc}})$ would of course equal N_{proc}, and at the very least, it should be greater than 1. A related measure, parallel efficiency, is defined as

$$PE(N_{\text{proc}}) = \frac{S(N_{\text{proc}})}{N_{\text{proc}}}. \tag{10}$$

Obviously the ideal value of $PE(N_{\text{proc}})$ is 1, and if $PE(N_{\text{proc}})$ is less than $1/N_{\text{proc}}$, it means that the simulation performed on N_{proc} cores performs worse than a simulation on a single processor.

Figures 15 and 16 show parallel efficiencies and wall clock times for the client application of the "KMCThinFilm" library implementing the coarsening model running on a Cray XC40 cluster of 2.3-GHz Intel Xeon E5-2698 v3 nodes, where each node effectively has 32 cores. Here, the model is applied over domains with in-plane lattice dimensions of 256×256, 512×512, $1{,}024 \times 1{,}024$, and $1{,}600 \times 1{,}600$. The parallel time-stepping scheme here uses a fixed time step set to D_0^{-1}. Both row-based and compact decomposition are used with the binary tree and Schulze solvers. In Fig. 15, the parallel efficiencies from the work of Shi, Shim, and Amar[38] are also shown for comparison, although this comparison should be taken with caution since their work was done on an entirely different cluster. From Figs. 15 and 16, certain trends are evident. One trend is that the use of the solver based on the Schulze algorithm leads to slightly smaller parallel efficiencies. This appears, though, to be an artifact of this solver simply taking far less computational time than the binary tree solver for the same problem, while not affecting the MPI communication time. A second trend is that while parallel efficiencies steadily decrease with the number of processing cores, the rate of this decrease is lower for larger domain sizes. For the $1{,}600 \times 1{,}600$ domain, parallel efficiencies are even sometimes marginally greater than 1. Such behavior has been seen before in the work of Shi, Shim, and Amar, and it may be due to memory cache effects.[38] A third trend is that for $N_{\text{proc}} > 1$, compact decomposition consistently leads to both lower parallel efficiencies and higher wall clock times than row-based decomposition, though as the size of the domain increases, the relative difference in wall clock

times for the 2 modes of decomposition decreases. For the 256×256 domain, this trend is so strong that when compact decomposition is used, parallel efficiencies are at or below $1/N_{\mathrm{proc}}$, which again means that the simulations performed on multiple processing cores performed at the same speed or worse than a simulation on a single processor. Accordingly, for this domain, the wall clock times for $N_{\mathrm{proc}} > 1$ are also generally greater than that for a single processing core, showing just how poorly compact decomposition performs over such a small domain. The relatively poor performance of compact decomposition versus row-based decomposition may seem surprising at first, given that compact decomposition should usually entail a lower volume of data being communicated between MPI processes. However, compact decomposition requires twice as many sectors as row-based decomposition, and each visit of a sector requires the communication of ghosts. The volume of data communicated in row-based decomposition may indeed be higher than that in compact decomposition. However, since only changes to ghost regions are communicated, this volume is not that high to begin with, so the communication overhead is dominated more by the very acts of sending and receiving messages, rather the costs associated with the volume of data itself. Shim and Amar[35] have also observed row-based communication having a lower overhead.

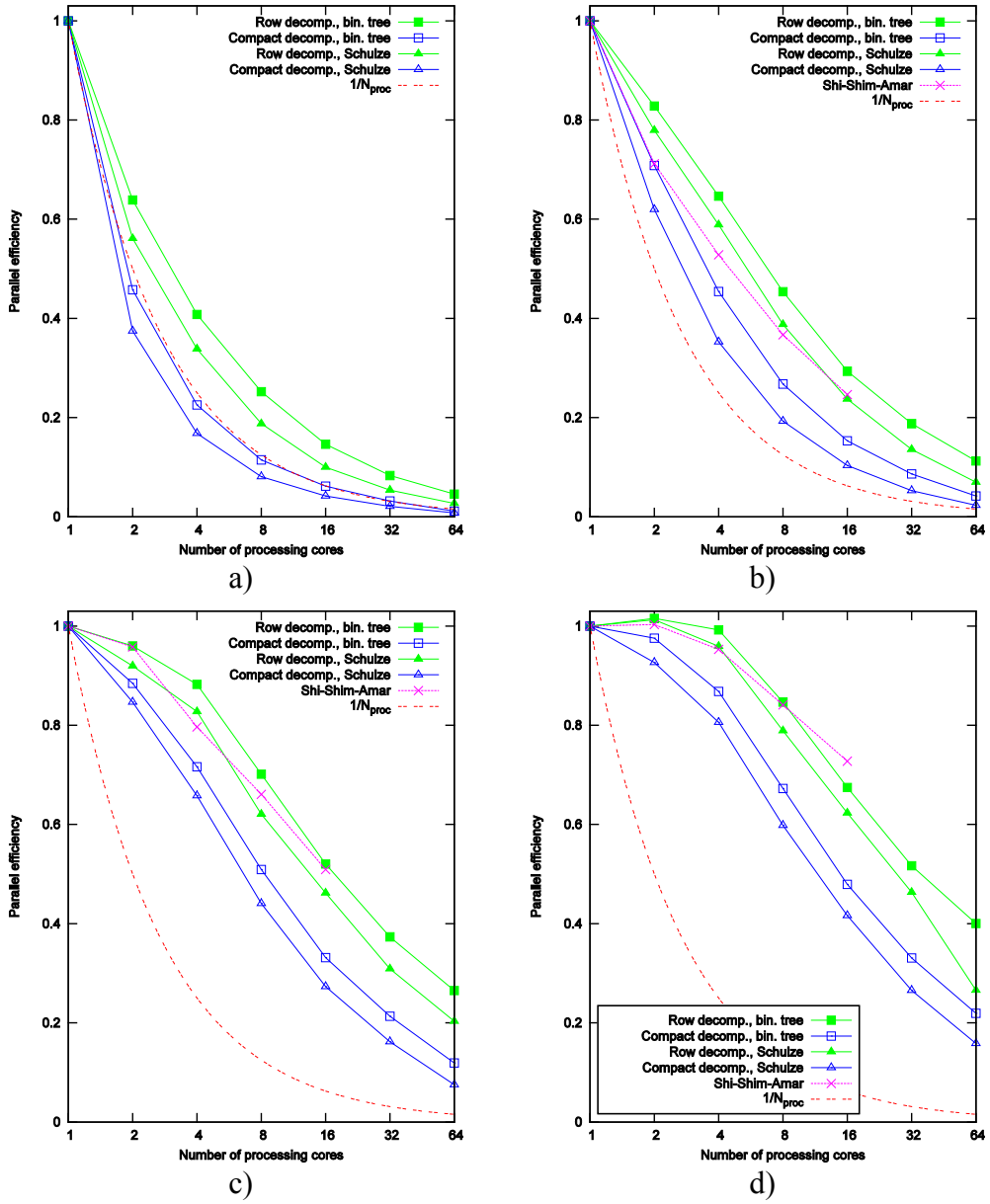


Fig. 15 Parallel efficiency versus the number of processing cores for the implementation of an island coarsening model using the "KMCThinFilm" library on a Cray XC40 cluster. The model is applied over domains with in-plane dimensions of a) 256 × 256, b) 512 × 512, c) 1,024 × 1,024, and d) 1,600 × 1,600. The parallel time-stepping scheme here uses a fixed time step set to $1/D_0$. Both row-based and compact decomposition are used with the binary tree and Schulze solvers. For models with in-plane dimensions from 512 × 512 to 1,600 × 1,600, parallel efficiencies from the island coarsening implementation of Shi, Shim, and Amar[38] (running on a different cluster) are also shown.

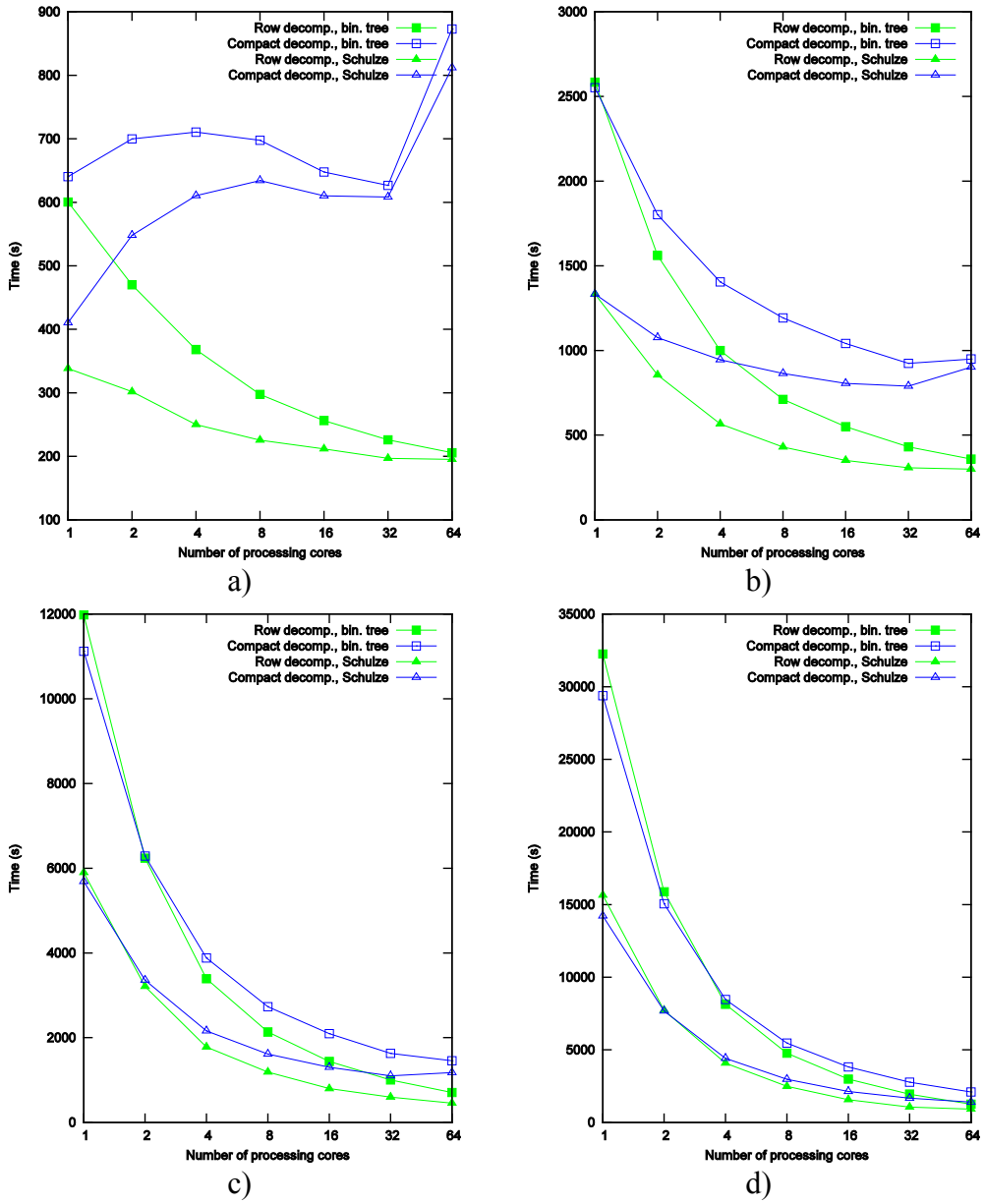


Fig. 16 Wall clock time versus the number of processing cores for the implementation of an island coarsening model using the "KMCThinFilm" library on a Cray XC40 cluster. The model is applied over domains in-plane dimensions of a) 256 × 256, b) 512 × 512, c) 1,024 × 1,024, and d) 1,600 × 1,600. The parallel time-stepping scheme here uses a fixed time step set to $1/D_0$. Both row-based and compact decomposition are used with the binary tree and Schulze solvers.

At this point, the time step for the parallel algorithm has been set to a fixed value. Here, possible consequences of using an adaptive time-step scheme on parallel efficiency are shown. For the client application of the "KMCThinFilm" library implementing the island coarsening model for the different parallel time-step schemes described in Section 2, Figs. 17 and 18 show the parallel efficiency, wall

clock times, average value of adaptive parallel time step t_{stop}, and the number of parallel time steps (i.e., the loops over sectors). For the fixed time-step scheme, the step size is again D_0^{-1}. The Schulze solver and row-based parallel decomposition are used. For all the adaptive schemes, the island coarsening model presents some problems. As expected, the maximum single propensity scheme generally leads to the same time step as the fixed time-step scheme with a time step of D_0^{-1}, but at a greater cost, both due to the computation needed to find the local maximum propensity values and to the cost of communication needed to find the global maximum propensity among all the MPI processes. For the 512×512 domain, the wall clock time at $N_{proc} = 64$ is higher than that at $N_{proc} = 32$, and for the 256×256 domain, there is no benefit in using more than 8 processing cores. The maximum average propensity scheme with $n_{stop} = 1$ has improved parallel efficiency, but for this particular model that introduces so-called "shish kebab" artifacts as shown in Fig. 19, where in a coarsening simulation on a 512×512 domain distributed over 8 processing cores, particles with no neighbors or only 1 neighbor line up near sector boundaries. This issue can be ameliorated by decreasing n_{stop} to 0.1. When the maximum average propensity scheme is used with this particular coarsening model, the parallel time step decreases as the number of processing cores increases. For the 256×256 domain, this effect is so acute that it ruins the parallel scalability for $N_{proc} > 4$ when $n_{stop} < 1$, and the number of parallel steps (i.e., the number of loops over sectors) can be more than that from the conservative maximum single propensity scheme. This happens because even after islands form, there are still a few particles with no lateral nearest neighbors, and because of the exponential function in the propensity of diffusion, these few particles have a propensity to diffuse that is several orders of magnitude higher than those of other particles. When the partitions are few and large, these few high-magnitude propensities are averaged among a large number of propensities, so that the average propensity of each sector is not dominated by them, and thus the parallel time step is not too small (relative to D_0^{-1}). As the number of partitions increases and their size decreases, eventually the average propensities in each sector become dominated by the high-magnitude propensities of those particles, causing the parallel time step to plummet. This issue is not present in systems where the available propensities do not vary by orders of magnitude, such as the Potts model of grain growth that was used to first demonstrate this type of adaptive scheme.[25,36]

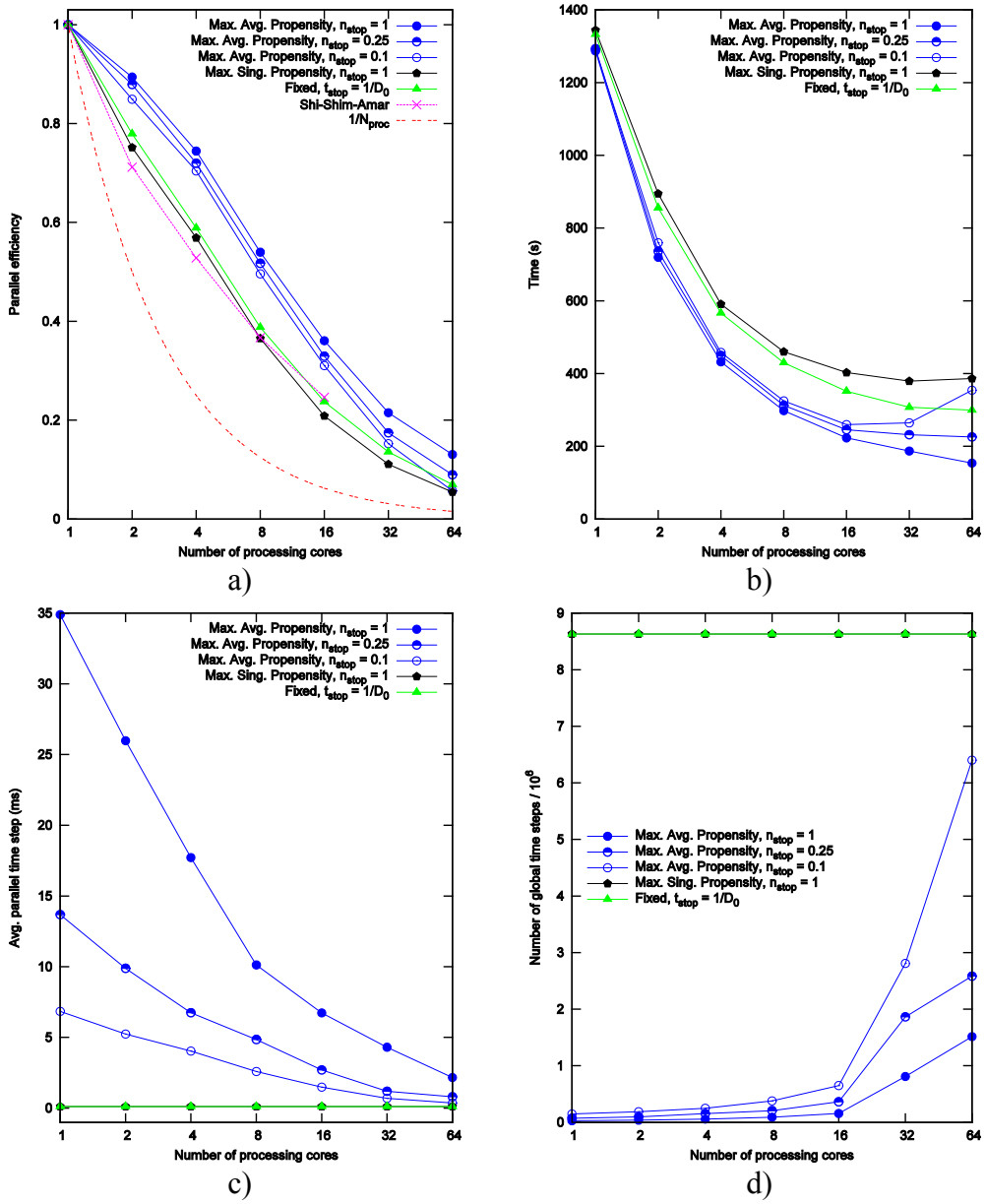


Fig. 17 Effects of the choice of adaptive time-step scheme on the parallel scalability of the implementation of an island coarsening model on a 512 × 512 domain, using the "KMCThinFilm" library on a Cray XC40 cluster. These figures show a) strong parallel scalability, b) wall clock time, c) average value of the adaptive time step, and d) the number of parallel time steps (i.e., the loops over sectors). "Max. Avg. Prop", "Max. Sing. Prop." and "Fixed" represent different parallel time-stepping schemes. These were determined using the solver based on the Schulze algorithm with row-based decomposition.

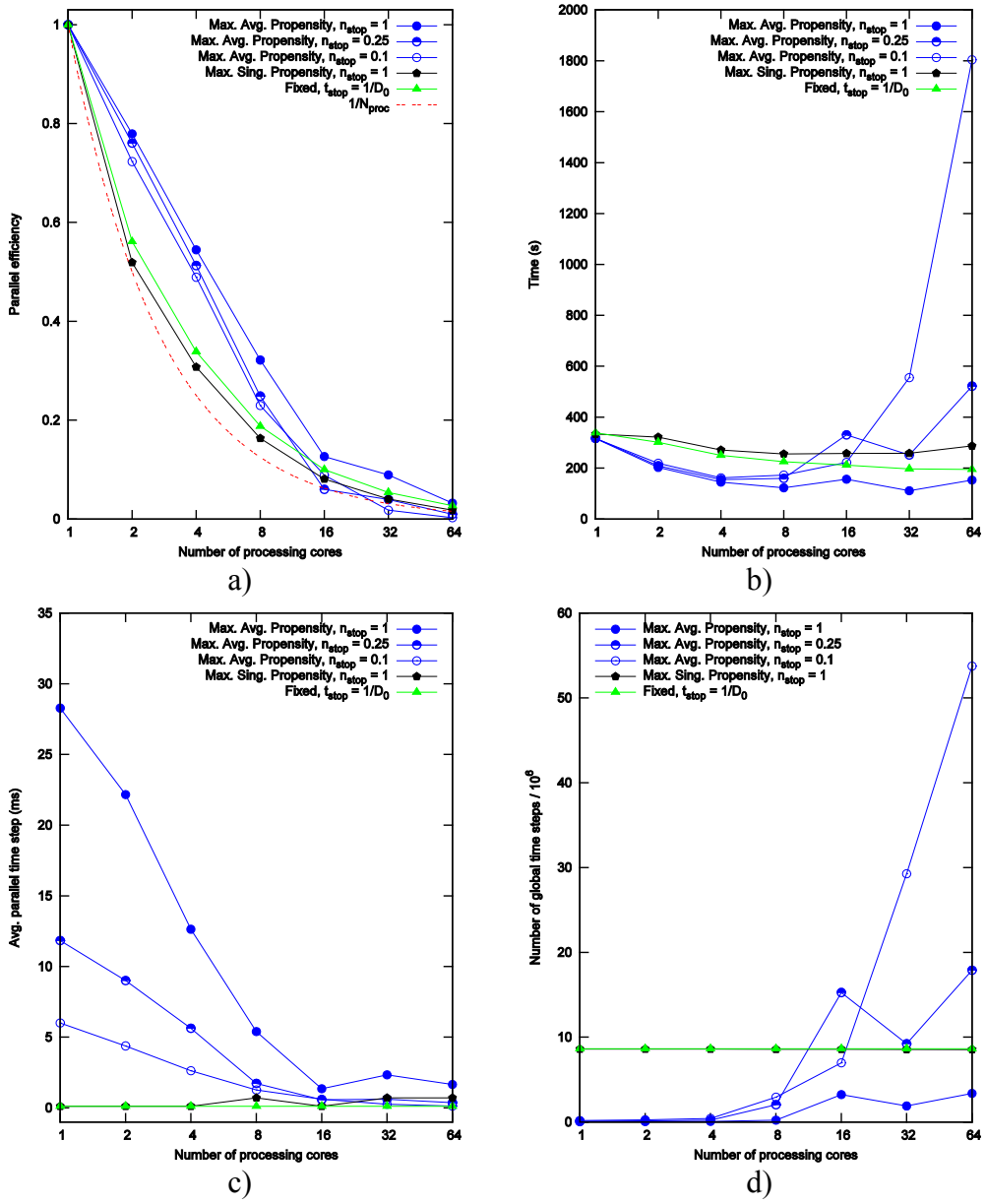


Fig. 18 Effects of the choice of adaptive time-step scheme on the parallel scalability of the implementation of an island coarsening model on a 256 × 256 domain, using the "KMCThinFilm" library on a Cray XC40 cluster. These figures show a) strong parallel scalability, b) wall clock time, c) average value of the adaptive time step, and d) the number of parallel time steps (i.e., the loops over sectors). "Max. Avg. Prop", "Max. Sing. Prop." and "Fixed" represent different parallel time-stepping schemes. These were determined using the solver based on the Schulze algorithm with row-based decomposition.

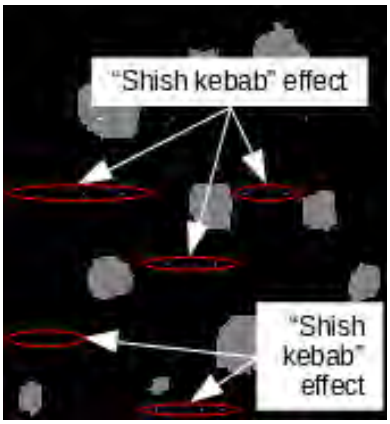


Fig. 19 Close-up of the results of an island coarsening simulation after 10^3 s of simulation time, on a 512 × 512 domain distributed over 8 processing cores, using the maximum average propensity adaptive parallel time-stepping scheme with n_{stop} = 1, the Schulze solver, and row-based parallel decomposition. These results show a "shish kebab" effect, where features of the simulation (in this case, particles with no neighbors or only 1 neighbor) are artificially attracted to sector boundaries due to approximations in the parallel kMC method.[36]

5. Conclusions

A new library for the development of lattice-based kMC simulation codes, "KMCThinFilm", has been introduced. This library implements algorithms for choosing sequences of random events in a kMC simulation, allowing researchers to focus on writing the code needed to determine the propensities for these events and the means of executing them. Example applications, 1 of a patterned substrate and 1 of ballistic deposition of a pseudo-polycrystalline film, have demonstrated its flexibility. The "KMCThinFilm" library is capable of approximate parallel kMC simulations, and its scalability appears to be on par with a previous parallel implementation. However, scalability still tends to decline fairly rapidly as more processing cores are used in a simulation, and this tendency is more pronounced for simulations that use smaller lattices. Furthermore, care needs to be taken in the determination of the time step used to determine the frequency of parallel synchronization used in a simulation. Even schemes that aim to determine this time step adaptively may lead to steps so large that they lead to artifacts or—depending on the system being modeled—so small that employing parallelism provides either a marginal performance benefit or no benefit over a purely serial simulation. Parallel simulations employing the "KMCThinFilm" library, then, may be best reserved for simulations that are too computationally expensive to run on a workstation.

6. References and Notes

1. Voter AF. Introduction to the kinetic Monte Carlo method. in Radiation Effects in Solids, Dordrecht, Netherlands, Springer, 2007, pp. 1–23.
2. Barnett SA, Rockett A. Monte Carlo simulations of Si(001) growth and reconstruction during molecular beam epitaxy. Surface Science. 1988;198:133–150.
3. Clarke S, Wilby MR, Vvedensky DD. Theory of homoepitaxy on Si(001). Surface Science. 1991;255:91–110.
4. Itoh M, Bell GRAAR, Jones TS, Joyce BA, Vvedensky DD. Island nucleation and growth on reconstructed GaAs(001) surfaces. Physical Review Letters. 1998;81(2):633–636.
5. Kuronen A, Nurminen L, Kaski K. Computer simulation of nucleation on patterned substrates. in MRS Fall Meeting, Boston (MA), 1999.
6. Wang L, Clancy P. Kinetic Monte Carlo simulation of the growth of polycrystalline Cu films. Surface Science. 2001;473:25–28.
7. Volkmann T, Ahr M, Biehl M. Kinetic model of II-VI(001) semiconductor surfaces: growth rates in atomic layer epitaxy. Physical Review B. 2004;69:165303.
8. Fu K, Fu Y, Han P, Zhang Y, Zhang R. Kinetic Monte Carlo study of metal organic chemical vapor deposition growth dynamics of GaN thin film at microscopic level. Journal of Applied Physics. 2008;103:103524.
9. Günther V, Mauss F, Klauer C, Schlawitschek C. Kinetic Monte Carlo simulation of the epitaxial growth of Si(100). Phys. Status Solidi. 2012;9(10-11):1955–1962.
10. Chen S, Liang J, Luo D, Jiang S. Onset of shadowing-dominated growth of Ag films in glancing angle deposition: Kinetic Monte Carlo simulation. Applied Surface Science. 2013;264:552–556.
11. Blue JL, Beichl I, Sullivan F. Faster Monte Carlo simulations. Physical Review E. 1995;51(2):R867–R868.
12. Schulze TP. Kinetic Monte Carlo simulations with minimal searching. Physical Review E. 2002;65:036704.
13. Hoffmann MJ, Matera S, Reuter K. kmos–a lattice kinetic Monte Carlo framework. Computer Physics Communications. 2014;2138-2150.

14. Lukkien JJ. CARLOS. 2009. Welcome to the CARLOS Project; [accessed 2014 Aug 8]. http://carlos.win.tue.nl.

15. Lukkien JJ, Segers JPL, Hilbers PAJ, Gelten RJ, Jansen APJ. Efficient Monte Carlo methods for the simulation of catalytic surface reactions. Physical Review E. 1998;58(2):2598–2610.

16. Dooling DJ. 2004. CHIMP HIerarchical Modeling Program; [accessed 2014 Aug 28]. http://chimp.sourceforge.net/.

17. Dooling DJ, Broadbelt, LJ. Generic Monte Carlo tool for kinetic modeling. Industrial and Engineering Chemistry Research. 2001;40(2):522–529.

18. Meekes H. Monty: kinetic Monte Carlo simulation package. 30 November 2004; [accessed 2014 Aug 28]. http://www.vsc.science.ru.nl/deij/monty.html.

19. Boerrigter SXM, Josten GPH, van de Streek J, Hollander FFA, Los J, Cuppen HM, Bennema P, Meekes H. MONTY: Monte Carlo crystal growth on any crystal structure in any crystallographic orientation; application to fats. J. Phys. Chem. A. 2004;108(27):5894–5902.

20. Vlachos Research Group: Multiscale Reaction Engineering; [accessed 2014 Aug 28]. http://dion.che.udel.edu/downloads/.

21. Stamatakis M, Vlachos DG. A graph-theoretical kinetic Monte Carlo framework for on-lattice chemical kinetics. J. Chem. Phys. 2011;134:214115.

22. Hoffmann MJ. kmos: A general lattice kinetic Monte Carlo simulation framework; [accessed 2014 Aug 28]. http://mhoffman.github.io/kmos/.

23. Kiravittaya S, Schmidt OG. Quantum-dot crystal defects. Applied Physics Letters. 2008;93:173109.

24. Leetmaa M, Skorodumova NV. KMCLib: A general framework for lattice kinetic Monte Carlo simulations. Computer Physics Communications. 2014;85:2340–2349.

25. Sandia National Laboratories. SPPARKS Kinetic Monte Carlo simulator; [accessed 2014 Aug 28]. http://spparks.sandia.gov/.

26. Wright SA, Plimpton SJ, Swiler TP, Fye RM, Young MF, Holm EA. Potts-model grain growth simulations: Parallel algorithms and applications. Sandia National Laboratories, Albuquerque (NM), 1997.

27. Here, a function object is an instance of a class that defines an overload for the "()" operator. It can be called as if it were a function, but can also contain other data members that may be used for parameters that do not change from call to call, pointers to other objects, or other things.

28. Meixner M, Kunert R, Schöll E. Control of strain-mediated growth kinetics of self-assembled quantum dots. Physical Review B. 2003;67:195301.

29. Here, "array" simply refers to a data structure with a number of elements N_e, each of which can be randomly accessed via an integer index $i \in [i_0, i_0 + N_e)$. There is no assumption here that N_e is fixed or that the elements are stored contiguously in memory. For example, in the C++ Standard Template Library, either a vector or a deque may effectively act as such an array.

30. Matsumoto M, Nishimura T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation. 1998;8(1):3–30.

31. L'Ecuyer P, Simard RJ, Chen EJ, Kelton WD. An object-oriented random-number package with many long streams and substreams. Operations Research. 2002;50(6):1073–1075.

32. Leydold J. RngStreams–multiple independent streams of pseudo-random numbers; 2009 January [accessed 2014 Sep 16]. http://statmath.wu.ac.at/software/RngStreams/.

33. Matsumoto M, Nishimura T. Dynamic Creation of Pseudorandom Number Generators. in Monte Carlo and Quasi-Monte Carlo Methods, Claremont (CA), 1998.

34. Matsumoto M, Nishimura T. Dynamic Creator Home Page; 2014 June 30 [accessed 2014 Sep 16]. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/DC/dc.html.

35. Shim Y, Amar JG. Semirigorous synchronous sublattice algorithm for parallel Kinetic Monte Carlo simulation of thin film growth. Physical Review B. 2005;71:125432.

36. Plimpton S, Battaile C, Chandross M, Holm L, Thompson A, Tikare V, Wagner G, Webb E, Zhou X, Cardona CG, Slepoy A. Crossing the mesoscale no-man's land via parallel kinetic Monte Carlo. Albuquerque (NM): Sandia National Laboratories (US); 2009 Oct. Report No.: SAND2009-6226. Also available at http://spparks.sandia.gov/papers.html.

37. Gropp W, Lusk E, Skjellum A. Using MPI portable parallel programming with the message-passing interface. 2nd ed. Cambridge (MA): MIT Press; 1999 Nov.

38. Shi F, Shim Y, Amar JG. Parallel kinetic Monte Carlo simulations of two-dimensional island coarsening. Physical Review E. 2007;76:031607.

39. Note that in this later paper, the prefactor for p_{hop} is given as $2k_BT/h$ rather than k_BT/h.

40. Nurminen L, Kuronen A, Kaski K. Kinetic Monte Carlo simulation of nucleation on patterned subtrates. Physical Review B. 2000;63:035407.

41. Levine SW, Clancey P. A simple model for the growth of polycrystalline Si using the kinetic Monte Carlo simulation. Modelling and Simulation in Materials Science and Engineering. 2000;8:751–762.

42. Shakarji CM. Least-squares fitting algorithms of the NIST algorithm testing system. Journal of Research of the National Institute of Standards and Technology. 1998;103(6):633–641.

43. Tait RN, Smy T, Brett MJ. Modelling and characterization of columnar growth in evaporated films. Thin Solid Films. 1993;226:196–201.

44. Kratzer P. Monte Carlo and kinetic Monte Carlo methods–a tutorial. In: Grotendorst J, Attig N, Blügel S, Marx D, editors. Multiscale Simulation Methods in Molecular Sciences. Institute for Advanced Simulation, Forschungszentrum Jülich, Germany; 2009, p. 51–76.

List of Symbols, Abbreviations, and Acronyms

Θ_c	Coverage
$\mathbf{a}_i$	Primitive lattice vector
$\hat{\mathbf{x}}$	Unit vector along Cartesian x-axis
$\hat{\mathbf{y}}$	Unit vector along Cartesian y-axis
$\hat{\mathbf{z}}$	Unit vector along Cartesian z-axis
F_{ijk}	The 1-dimensional array at lattice cell (i,j,k) holding double precision floating-point values
F_{stop}	Some measure of the propensities in the part of the lattice owned by a given MPI process in a parallel kMC simulation that determines the parallel global time step t_{stop} when an adaptive time step is used. Which measure of propensities is used depends on the choice of adaptive time step scheme. $t_{stop} = n_{stop}/F_{stop}$.
I_{ijk}	The 1-dimensional array at lattice cell (i,j,k) holding integer values
N_{proc}	Number of MPI processes
k_B	Boltzmann's constant
n_{stop}	Prefactor to adjust parallel global time step t_{stop}
t_{stop}	Size of global time step in approximate parallel kMC simulation
ν_0	Attempt frequency
h	Planck's constant
F	Deposition flux
$PE(N)$	Parallel efficiency for N MPI processes
$S(N)$	Parallel speedup for N MPI processes
T	Temperature
a	Lattice constant
θ	Angle of incidence of a depositing particle
DCMT	Dynamic Creator of Mersenne Twister
fcc	face-centered cubic

kMC	kinetic Monte Carlo
ML	monolayer(s)
MPI	Message Passing Interface
SPPARKS	Stochastic Parallel PARticle Kinetic Simulator
XML	Extensible Markup Language

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

1 (PDF) DIR USARL RDRL CIH C J J RAMSEY